



UNIVERSIDAD NACIONAL  
AUTÓNOMA DE  
MÉXICO

**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**  
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

**“MODELADO DE BASES DE DATOS SEMI-ESTRUCTURADOS EN XML”**

**TESIS**  
**QUE PARA OPTAR POR EL GRADO DE:**  
**MAESTRA EN INGENIERÍA (COMPUTACIÓN)**

**PRESENTA:**  
**YESICA PAOLA DIEGO MOTA**

**DRA. AMPARO LÓPEZ GAONA**  
**FACULTAD DE CIENCIAS**  
**PROGRAMA DE POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN**

**MÉXICO, D. F. AGOSTO 2013**



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

## Agradecimientos

---

A la Dra. Amparo López Gaona, por guiarme durante la realización de este trabajo, por su amabilidad y disponibilidad en el tiempo que me dedicó para que este trabajo culminara

A mis padres y hermanas por alentarme a continuar con mi formación profesional y por brindarme todo su apoyo cada día de mi vida

A mis sinodales por leer este trabajo y hacer las recomendaciones que dieron más valor a su contenido

A mis compañeros de maestría, porque de ellos aprendí muchas cosas y tuve momentos muy agradables a su lado

## Resumen

---

A pesar de que XML no se concibió inicialmente como una tecnología que soportara Bases de Datos, debido a que, permite la representación y la integración de datos semi-estructurados y estructurados, ha dado paso al surgimiento de las bases de datos Nativas XML, en las cuales la información es tratada como una colección de documentos. Actualmente existen sistemas gestores para este tipo de Bases de Datos, sin embargo, estas bases de datos carecen de una metodología aceptada y aprobada para el diseño de las mismas, por lo que no se cuenta con ningún modelo conceptual que permita capturar los requerimientos de algún problema. Para solventar esta situación, en la presente tesis se propone un modelo para Bases de Datos semi-estructurados con XML, el cual se basa en el modelo entidad relación para bases de datos relacionales, esto con el objetivo de que sea fácil de entender y aplicar para los diseñadores; a si mismo se proporciona un listado de las buenas prácticas, las cuales deben tomarse en cuenta a la hora de modelar un problema que tenga que ver con datos semi-estructurados; finalmente, se da a conocer el conjunto de reglas que los diseñadores deben seguir para transformar el diagrama resultante del modelado a una DTD, que garantice que los documentos XML de la base de datos, son válidos y bien formados.

## Contenido

---

Agradecimientos .....	2
Resumen.....	3
Introducción .....	6
Capítulo 1. Conceptos Generales de XML .....	9
1.1 ¿Qué es XML? .....	9
1.2 Componentes de un Documento XML .....	10
1.2.1 Elementos.....	10
1.2.2 Atributos.....	11
1.2.3 Espacios de Nombres .....	12
1.2.4 Comentarios .....	13
1.2.5 Instrucciones de Procesamiento .....	13
1.3 Estructura de un Documento XML.....	13
1.4 Documentos XML Válidos y Bien-Formados .....	14
1.5 DTD (Definición de Tipo de Documento) .....	15
1.6 XML Schema .....	22
1.7 Tipos de Datos (Estructurados, No Estructurados y Semi-estructurados).....	25
1.8 Tipos de Documentos XML.....	27
Capítulo 2. Bases de Datos XML.....	29
2.1 La Importancia de las Bases de Datos .....	29
2.2 XML y Bases de Datos.....	30
2.3 Bases de Datos Habilitadas para XML .....	31
2.4 Bases de Datos Nativas XML .....	32
2.5 Lenguajes de Consulta.....	34
2.5.1 XPath .....	34
2.5.2 XQuery.....	37
2.6 Sistemas Gestores de Bases de Datos XML Nativos.....	39
2.6.1 Tamino.....	39
2.6.2 eXist.....	41

2.7	Uso de Bases de Datos XML Nativas .....	43
Capítulo 3. Diseño de Bases de Datos .....		45
3.1	Importancia del Diseño de una Base de Datos .....	45
3.2	Modelado de Datos .....	47
3.3	Modelos para Datos Semi-estructurados.....	48
3.3.1	DOM (Document Object Model) .....	49
3.3.2	OEM (Object Exchange Model) .....	51
3.3.3	S3-graph (Semi-Structured Schema Graph) .....	52
3.3.4	CM Hypergraph and SchemeTree .....	54
3.3.5	EER (Entity Relationship diagram) y XGrammar.....	56
3.4	Diseño de Bases de Datos XML .....	60
Capítulo 4. Modelo de Base de Datos Semi-estructurados propuesto .....		61
4.1	Notación Gráfica.....	78
4.2	Buenas prácticas aplicables al modelo.....	80
Capítulo 5. Reglas de transformación del modelo a una DTD.....		82
(Definición de Tipo de Documento) .....		82
Capítulo 6. Modelado de sistema: Librería .....		95
6.1	Definición del Problema .....	95
6.2	Desarrollo del Diagrama.....	96
6.2.1	Modelo de Base de datos: Librería.....	101
6.3	Traducción del modelo a la DTD (Definición de Tipo de Documento).....	102
6.3.1	DTD (Definición de Tipo de Documento) para la Base de Datos: Librería .....	110
Conclusiones .....		112
Referencias .....		114

## Introducción

---

El intercambio de información siempre ha sido un problema cuando se utilizan lenguajes y sistemas operativos distintos, esto, se ha dado desde los niveles de procesadores de palabras, hasta la utilización de bases de datos que manejan diferentes tipos de datos.

XML (eXtensible Markup Language) es un conjunto de reglas para definir etiquetas semánticas que dividen un documento en partes e identifica las diferentes partes del documento. XML actualmente se ha convertido en un estándar para compartir e intercambiar información.

XML es una representación común de datos semi-estructurados, los cuales, no tienen una estructura estrictamente definida, este tipo de datos pueden encontrarse por ejemplo, en aplicaciones y servicios web; debido al incremento de uso de este tipo de datos en la web, los datos semi-estructurados necesitan ser modelados, almacenados y manipulados apropiadamente para su efectiva utilización; a pesar de que, XML inicialmente no se concibió como una tecnología utilizada en aplicaciones de bases de datos, puede representar datos semi-estructurados y estructurados utilizados en aplicaciones de negocios(SILBERSCHATZ, 2006), debido a esto, XML dio paso a la aparición de una nueva generación de Bases de Datos, conocidas como: “Bases de Datos Nativas XML”(CORNEJO, 2002), las cuales permiten integrar datos estructurados y semi-estructurados.

### Problema

Un sistema de base de base de datos bien definido se basa en un modelo de datos bien definido(SALMINEN, 2001). La complejidad de los repositorios de datos relacionados con XML y la necesidad de integrar el manejo de documentos estructurados con el manejo de otros tipos de datos crea un reto especial para el modelo de datos subyacente de los documentos XML, siendo este, un DTD o un XML Schema. En muchos artículos de investigación modelar datos XML da como resultado un árbol etiquetado o a un grafo dirigido; este tipo de modelado puede ser suficiente para el desarrollo de las capacidades relativas a la estructura jerárquica de los elementos. Sin embargo, para ser capaz de manejar documentos XML como una base de datos, se requiere un modelo de datos más rico.

En la actualidad el diseño de bases de datos XML no tiene una metodología tan aceptada como para el relacional(MORO, 2009), por lo que, no se cuenta con un modelo conceptual que permita capturar las necesidades del sistema. Una base de datos XML es generalmente modelada a través de árboles, donde un nodo corresponde a un elemento, atributo o valor y una arista representa relaciones elemento-subelemento o elemento-valor. El diseño de estas bases de datos sería sencillo si se pudiera aplicar el modelo Entidad Relación o Relacional directamente, sin embargo, XML es semi-estructurado y con características muy flexibles, lo que hace que estos modelos sean inservibles. Algunas razones son: en XML los datos son inherentemente ordenados, permite diferentes estructuras y tipos complejos, como opcionales y elementos multivalorados, no hay una manera simple de especificar relaciones muchos a muchos, esto, por la estructura jerárquica del XML.

## Contribuciones

Debido a que actualmente no hay un modelo aceptado para representar formalmente el diseño de las bases de datos semi-estructurados con XML. La principal contribución de la presente tesis es proveer a los diseñadores de bases un modelo gráfico, que permita capturar los requerimientos del problema, así como buenas prácticas que ayuden al diseñador a tomar decisiones a la hora de modelar un problema; este modelo permite capturar datos semi-estructurados y estructurados; por otra parte, se establece una serie de reglas que permiten realizar la traducción del modelo conceptual al modelo subyacente de documentos XML una DTD.

## Recorrido por el resto de la Tesis

La tesis está organizada en cinco capítulos: los primeros capítulos tratan sobre conceptos de XML, bases de datos con XML y diseño de bases de datos; el modelo propuesto para el diseño de bases de datos semi-estructurados y el conjunto de reglas de transformación del modelo a una DTD, se dan a conocer en los capítulos siguientes; se dedica un capítulo al desarrollo de un ejemplo, aplicando el modelo a un problema y obteniendo su DTD; finalmente se dan a conocer las conclusiones, trabajo futuro y referencias. El contenido de los capítulos es el siguiente:



**Capítulo 1. Conceptos Generales de XML.** El capítulo presenta los conceptos básicos sobre XML, dando a conocer la manera en cómo se estructuran los documentos XML; los tipos de datos soportados por XML, los tipos de documentos y las alternativas de validación.

**Capítulo 2. Bases de Datos XML.** En este capítulo se da a conocer la relación de la tecnología XML con las bases de datos, así como los tipos de bases de datos XML existentes; se describen los lenguajes de consulta de las bases de datos con XML, se proporciona la definición de Bases de Datos Nativas con XML y la descripción de dos de los sistemas gestores de bases de datos nativas más populares; por último se dan a conocer las áreas donde comúnmente son aplicadas las bases de datos XML.

**Capítulo 3. Diseño de Bases de Datos.** En este capítulo se resalta la importancia del diseño de las bases de datos y se dan a conocer los modelos existentes para el diseño de bases de datos semi-estructurados.

**Capítulo 4. Modelo de Base de Datos Semi-Estructurados.** En este capítulo se presenta el modelo propuesto en la tesis con sus elementos, así como, la notación gráfica y las buenas prácticas aplicables al modelo.

**Capítulo 5. Reglas de transformación del modelo a una DTD.** En este capítulo se dan a conocer una serie de reglas que permiten obtener la DTD del documento desde un diagrama resultante de la aplicación del modelo propuesto en el capítulo 4.

**Capítulo 6. Modelado del Sistema.** En este capítulo se realiza el modelado de un problema aplicando el modelo propuesto en el capítulo 4 y se genera la DTD utilizando las reglas de transformación del capítulo 5.

**Conclusiones.** En este apartado se dan a conocer las conclusiones, aportaciones y el trabajo futuro referente al modelo propuesto.

# Capítulo 1. Conceptos Generales de XML

---

## 1.1 ¿Qué es XML?

XML<sup>1</sup> significa Lenguaje de Marcas Extensible, del inglés eXtensible Markup Language es un metalenguaje que proporciona una manera sencilla de definición de lenguajes de etiquetas estructuradas, en otras palabras, XML define un conjunto de reglas semánticas que permiten la organización de información de distintas maneras, XML tuvo sus inicios en un lenguaje creado por IBM en los años 70's, llamado inicialmente GML<sup>2</sup>, el cual fue normalizado por la ISO en 1986, creando así el SGML<sup>3</sup>, que es un lenguaje descriptivo, para estructurar documentos; este ofrecía nombres para categorizar e identificar partes de un documento, siendo capaz de solucionar una gran cantidad de problemas con la introducción de:

- El uso de etiquetas de apertura y cierre para que los navegadores fueran capaces de interpretar y dar formato al texto incluido en ellas.
- La posibilidad del uso de enlaces internos y externos a otros documentos, incrementando la posibilidad de navegación.

SGML fue principalmente utilizando en el intercambio, gestión y publicación de documentos, su nivel de complejidad impidió su aprobación en distintos ambientes de aplicación como World Wide Web. Fue así como surgió el estándar XML, que tenía como propósito mantener el poder y flexibilidad de SGML, pero sin la complejidad de éste (AGUILAR, 2010).

El desarrollo de XML comenzó en 1996 y desde entonces ha tenido un crecimiento exponencial. En realidad, surge del campo empresarial, para dotar a la WEB o a una red corporativa de más funcionalidad, y acceder a la información de una forma más flexible y adaptable. En febrero de 1998 fue adoptado como recomendación por el World Wide Web Consortium (W3C) Organismo que vela por el desarrollo de Internet y sus estándares (LAMARCA, 2011).

La primera definición de XML fue: "Sistema para definir, validar y compartir formatos de documentos en la Web" (LAMARCA, 2011). La diferencia fundamental con HTML<sup>4</sup> es que éste está

---

<sup>1</sup> Acrónimo de eXtensible Markup Language

<sup>2</sup> Acrónimo de Generalized Markup Language

<sup>3</sup> Acrónimo de *Standard Generalized Markup Language*

<sup>4</sup> Acrónimo de HyperText Markup Language

orientado a la presentación de datos, mientras que, XML está orientado a los datos en sí mismos, es extensible y por lo tanto es susceptible a ser ampliado o modificado por el cambio o la adición de características. XML puede tener etiquetas de su propia creación que son únicas para cada documento creado(POWELL, 2007);sin duda, esta diferencia es fundamental para los nuevos desarrollos de la Web donde se da suma importancia al contenido de los datos y a su tratamiento, y no sólo a su presentación(LAMARCA, 2011), actualmente XML es considerado un lenguaje estándar para representación e intercambio de datos, principalmente en Internet, debido a que es particularmente útil como formato de datos cuando dos aplicaciones se deben comunicar entre sí o integrar información de varias aplicaciones(SILBERSCHATZ, 2006).

Para la familia de lenguajes de marcado, en los que se incluyen HTML, SGML Y XML, las etiquetas describen el significado y la estructura de los datos, de la forma <etiqueta> y </etiqueta>, delimitando el comienzo y final de la porción de documento a la cual se refiere la etiqueta, en XML no se imponen las etiquetas permitidas, y se pueden elegir las que se consideren necesarias para cada aplicación, ésta característica es la clave de la función principal de XML en la representación e intercambio de datos(SILBERSCHATZ, 2006).

Entre las ventajas se mencionan: las etiquetas hacen que el texto en un documento XML sea autodocumentado, el formato del documento no es rígido y que permite estructuras anidadas, la heterogeneidad en donde cada instancia de datos puede contener diferentes campos de datos, similar al mundo real; la extensibilidad, en donde nuevos tipos de datos pueden ser agregados; y la flexibilidad, en donde los campos pueden cambiar de tamaño y configuración de instancia a instancia(BARRIOS, 2009).

## 1.2 Componentes de un Documento XML

### 1.2.1 Elementos

El constructor fundamental en un documento XML es el elemento. Un elemento es un conjunto de datos del documento, delimitado por un par de etiquetas de inicio y finalización coincidentes, y todo el texto que aparece entre ellas. Los documentos deben tener un único elemento raíz que abarque al resto de elementos del documento; los elementos de un documento se deben anidar adecuadamente, aunque, el anidamiento adecuado es una propiedad intuitiva, es necesario

definirla más formalmente; se dice que el texto aparece en el contexto de un elemento si aparece entre las etiquetas de inicio y finalización de dicho elemento; las etiquetas están anidadas adecuadamente si toda etiqueta de inicio tiene una etiqueta de finalización coincidente que está en el contexto del mismo elemento padre. La Figura 1.1 ilustra el uso de los elementos cuenta, numero-cuenta, nombre-sucursal y saldo.

```
...
<cuenta>
  Esta cuenta se usa muy rara vez por no decir nunca.
  <numero-cuenta> C-102 </numero-cuenta>
  <nombre-sucursal> Banco Central</nombre-sucursal>
  <saldo> 400 </saldo>
</cuenta>
...
```

**Figura 1.1** Uso de Elementos

### 1.2.2 Atributos

Además de elementos, XML especifica la noción de atributo; los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos del documento. Por ejemplo, el tipo de cuenta se puede representar como un atributo, como se ilustra en la Figura 1.2. Los atributos de un elemento aparecen como pares **nombre=valor** antes del cierre de una etiqueta y son cadenas que pueden aparecer sólo una vez en una etiqueta dada, al contrario de los subelementos que pueden estar repetidos.

En el contexto de construcción de un documento es importante la distinción entre un subelemento y un atributo, ya que un atributo es implícitamente texto que no aparece en el documento impreso.

```
...
<cuenta tipo-cuenta = nomina>
  <numero-cuenta> C-102 </numero-cuenta>
  <nombre-sucursal>Banco Central</nombre-sucursal>
  <saldo> 40000</saldo>
</cuenta>
...
```

**Figura 1.2** Uso de Atributos

### 1.2.3 Espacios de Nombres

Debido a que los documentos XML se diseñan para el intercambio de datos entre aplicaciones se tiene que introducir un mecanismo de espacio de nombres, que permita a las organizaciones especificar nombres únicos, que sean utilizados como marcas de los documentos. La idea de un espacio de nombres, es anteponer cada etiqueta o atributo un identificador de recursos universal (por ejemplo, una dirección Web). Tomando como ejemplo un sistema de control de sucursales de algún banco, si el Banco Principal deseara asegurar que los documentos creados no dupliquen las etiquetas usadas por los documentos de otros socios del negocio, se puede anteponer un identificador único con dos puntos a cada nombre de etiqueta. El uso de identificadores únicos largos en las etiquetas puede ser poco conveniente por lo que el espacio de nombres estándar proporciona una forma de definir una abreviatura para los identificadores (SILBERSCHATZ, 2006).

En la Figura 1.3 el elemento raíz (banco) tiene un atributo **xmlns:BP**, que declara que BP está definido como abreviatura para el URL dado. Se puede usar entonces la abreviatura en varias marcas de elementos como se ilustra en la figura. Un documento puede tener más de un espacio de nombres, declarado como parte del elemento raíz. Se pueden asociar entonces elementos diferentes con espacios de nombres distintos; se puede definir un espacio de nombres predeterminado mediante el uso del atributo xmlns en lugar de xmlns:BP en el elemento raíz.

Los elementos sin un prefijo de espacio de nombres explícito pertenecen entonces al espacio de nombres predeterminado.

```
<banco xmlns:BP = «http://www.BancoPrincipal.com»>
  ...
  <BP:sucursal>
    <BP:nombresucursal> Centro </BP:nombresucursal>
    <BP:ciudadsucursal> Brooklyn </BP:ciudadsucursal>
  </BP:sucursal>
  ...
</banco>
```

**Figura 1.3.** Uso de espacios de Nombres

#### 1.2.4 Comentarios

En ocasiones es necesario insertar comentarios en documentos XML, éstos son simplemente para fines informativos de la persona que está diseñando el documento y no serán parte de la salida impresa (AGUILAR, 2010).

Los comentarios pueden ser insertados en cualquier parte del documento, excepto dentro de las declaraciones de otros comentarios o de etiquetas. El formato para los comentarios en XML tiene la siguiente sintaxis inicia con la cadena “<!--” y termina con “-->”. En la Figura 1.4 se muestra un ejemplo del uso de comentarios.

```
<!--Inicia el Documento-->
<root>
.
.
.
</root>
```

**Figura 1.4.** Uso de Comentarios

#### 1.2.5 Instrucciones de Procesamiento

XML proporciona instrucciones de procesamiento, mediante las cuales las aplicaciones que leen el documento saben qué hacer con el mismo. Toda instrucción de procesamiento en XML inicia con “<?” y termina con “>” por ejemplo: <?xml-stylesheet ref=“empresa.css” type=“text/css” ?>. En este caso xml-stylesheet se usa para indicar al navegador que debe aplicar la hoja de estilo empresa.css antes de presentar este documento al usuario. Este tipo de instrucciones pueden ser incluidas en cualquier parte del documento XML, incluso antes o después del nodo raíz (AGUILAR, 2010).

### 1.3 Estructura de un Documento XML

Según (WILLIAMS, 2000), para que un documento XML se encuentre bien estructurado debe cumplir con las siguientes reglas:

1. Cada etiqueta de apertura debe tener su etiqueta de cierre correspondiente.

2. Los elementos en el documento, pueden contener subelementos pero no superponerse, por lo tanto el siguiente documento es incorrecto porque rompe esta regla:

```
<persona>  
<nombre>  
</nombre>  
<apellido>  
</persona>  
</apellido>
```

3. En todo documento XML, únicamente debe existir un elemento raíz.
4. Los valores de los atributos deben ser escritos entre comillas.
5. Los atributos para cada elemento deben ser únicos, es decir, un elemento no puede contener dos atributos con el mismo nombre.
6. Si en el documento existen comentarios o instrucciones de procesamiento, éstas no pueden formar parte de los elementos.
7. Los caracteres < o \$ no pueden ser parte de los datos en un atributo o elemento.

## 1.4 Documentos XML Válidos y Bien-Formados

A pesar de que la sintaxis de XML es flexible, está limitada por una gramática que rige el nombre de las etiquetas, el apego de los atributos a las etiquetas y así sucesivamente. Todos los documentos XML deberían ajustarse a las reglas básicas de la gramática, los documentos formados en base a estas reglas se dice que están bien formados y pueden ser interpretados por un intérprete XML, lo que significa, que no es necesario crear un intérprete para cada instancia de documento XML(B. Akmal Chaudhri., 2003).

Los documentos XML deben seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos.

En los sistemas de bases de datos relacionales se cuenta con esquemas que se usan para restringir qué información se puede almacenar en la base de datos y para restringir los tipos de datos de la información almacenada. En cambio los documentos XML se pueden crear de forma predeterminada sin un esquema asociado. Un elemento puede tener entonces cualquier subelemento o atributo. Aunque dicha libertad puede ser aceptable, algunas veces dada la naturaleza autodescriptiva del formato de datos no es útil, generalmente cuando los documentos

XML se deben procesar automáticamente como parte de una aplicación o incluso cuando se van a dar formato en XML a grandes cantidades de datos relacionados.

Además de que un documento XML debe estar bien formado, su estructura podría ser validada. La validez se refiere a que cada aplicación XML, debe verificar cuál es la relación que debe existir entre cada uno de los elementos que aparecen en el documento, dicha relación debe especificarse en un documento externo que bien puede ser una DTD<sup>5</sup> o un Esquema XML, porque éstos son los que van a indicar la estructura que tendrá internamente cada documento XML creado. Cabe aclarar que para el caso donde se incluyen ligaduras de integridad la DTD o el esquema aún son medios débiles para verificar que estas se cumplan (AGUILAR, 2010).

## 1.5 DTD (Definición de Tipo de Documento)

La necesidad de jerarquizar y estructurar correctamente la información no sólo para almacenarla, sino también para acceder a ella es de gran relevancia.

Inicialmente se usaron las Definiciones de Tipo de Documento en el lenguaje SGML para describir el vocabulario necesario para identificar todos los elementos de que iba a constar el documento y para expresar su estructura.

Una Definición de Tipo de Documento (DTD) desde una perspectiva de Bases de Datos proporciona un método de validación estructural de un documento XML, debido a que permite la definición de los componentes básicos del documento XML, es decir, es un conjunto de reglas sintácticas para definir elementos, indica qué elementos se pueden utilizar en un documento y el contenido de ellos, el orden en que deben aparecer los elementos, así como su anidamiento, los atributos que pueden tener, etc., una DTD nos permite crear un lenguaje propio de marcado para una aplicación específica; quien necesite usar XML para intercambio de datos debe definir su propia DTD.

La utilización de una DTD ayuda a que grupos independientes de personas puedan ponerse de acuerdo en tener un estándar para el intercambio de información. Las aplicaciones pueden usar un estándar DTD para verificar si la información que reciben es válida (W3Schools, 2012).

---

<sup>5</sup> Acrónimo de Document Type Definition



La DTD es una parte opcional de un documento XML. El propósito principal de un DTD es similar al de un esquema, restringir el tipo de información presente en el documento. Sin embargo no restringe en realidad los tipos en el sentido de tipos básicos como entero o cadena; en su lugar sólo restringe el aspecto de los subelementos y atributos en un elemento(SILBERSCHATZ, 2006).

La DTD puede estar contenida en el documento XML, como parte de su declaración de tipo de documento, aunque se suele crear aparte con la extensión “.dtd”, para posteriormente ser invocado en cualquier otro documento XML. Un ejemplo de la declaración de la DTD dentro del documento XML se muestra en la figura 1.5 que trata de un DTD acerca de notas:

```
<?xml version="1.0"?>
<!DOCTYPE nota [
<!ELEMENT nota (para, de, encabezado, cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT encabezado (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
]>
<nota>
<para>Juan</para>
<de>Emanuel</de>
<encabezado>Recordatorio</encabezado>
<cuerpo>No olvides la comida del fin de semana</cuerpo>
</nota>
```

**Figura 1.5** Ejemplo de declaración de la DTD dentro del documento

Si la DTD es declarada en un archivo aparte con extensión .dtd, se debe hacer una llamada al documento, en la figura 1.6 muestra un ejemplo de un documento XML donde hace la llamada a la DTD de nombre “nota.dtd”, la cual quedaría declarada de la misma manera que en la figura 1.5.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "nota.dtd">
<nota>
<para>Juan</para>
<de>Emanuel</de>
<encabezado>Recordatorio</encabezado>
<cuerpo>No olvides la comida del fin de semana</cuerpo>
</nota>
```

**Figura 1.6** Ejemplo de declaración de la DTD dentro del documento XML

Desde el punto de vista de una DTD un documento XML se conforma de los siguientes bloques de construcción:

- **Elementos**
- **Atributos**
- **Entidades**
- **PCDATA (Parsed Character Data).**
- **CDATA (Character Data).**

### **Elementos**

Los elementos representan nombres de etiquetas que pueden ser utilizadas en los documentos, las declaraciones de los elementos se realizan mediante el uso de <!ELEMENT>. Los elementos pueden contener subelementos o estar vacíos. Los elementos pueden tener atributos; la estructura de los subelementos está definida por un modelo de contenido construido de operadores aplicados a subelementos. Los elementos pueden estar agrupados por secuencias (a,b) o como opciones (a|b). Para todos los elementos o grupo de elementos, el modelo de contenido puede especificar su ocurrencia a través del uso de operadores de expresiones regulares (? , \* , +). Existe también un caso especial en éste modelo, el elemento EMPTY es un tipo de elemento que no tiene subelementos; ANY es un tipo de elemento que puede contener algún subelemento y #PCDATA es utilizado cuando un elemento que puede contener únicamente texto. Cuando el elemento puede contener subelementos mezclados con texto el modelo de contenido es llamado contenido mixto(SAHUGUET, 2001).

A continuación se listan los tipos de contenido de los elementos y la sintaxis de los mismos:

**Elemento Vacío**, como se ha mencionado anteriormente no contiene subelementos ni texto.

Sintaxis:

```
<!ELEMENT nombre_elemento EMPTY>
```

Ejemplo:

```
<!ELEMENT br EMPTY>
```

### **Cualquier tipo de contenido.**

Sintaxis:

```
<!ELEMENT nombre_elemento ANY>
```

Ejemplo:

```
<!ELEMENT note ANY>
```

### **Sólo texto.**

Sintaxis:

```
<!ELEMENT nombre_elemento (#PCDATA)>
```

Ejemplo:

```
<!ELEMENT para (#PCDATA)>
```

**Otros elementos.** Un elemento puede contener varios elementos a su vez, se puede declarar una lista de alternativas u opciones, con esto se podrá elegir una opción para que aparezca como subelemento del elemento principal, esta declaración se realiza mediante el pipe "|". Para definir la secuencia de subelementos que puede tener un elemento se utiliza la coma "," significando el orden en que deben aparecer los elementos en el documento XML.

### **Secuencia.**

Sintaxis:

```
<!ELEMENT nombre_elemento (hijo1,hijo2,...)>
```

Ejemplo:

```
<!ELEMENT nota (para, de, encabezado, cuerpo)>
```

### **Listado de opciones.**

Sintaxis:

```
<!ELEMENT nombre_elemento (hijo1|hijo2|...)>
```

Ejemplo:

```
<!ELEMENT nota (para|de|encabezado|cuerpo)>
```

En una declaración completa los subelementos deben ser declarados, la declaración completa del elemento "nota" que se ha utilizado en los ejemplos anteriores es la siguiente:

```
<!ELEMENT nota (para, de, encabezado, cuerpo)>  
<!ELEMENT para (#PCDATA)>  
<!ELEMENT de (#PCDATA)>  
<!ELEMENT encabezado (#PCDATA)>  
<!ELEMENT cuerpo (#PCDATA)>
```

### **Contenido Mixto.** Otros elementos y texto.

Ejemplo:

```
<!ELEMENT note (#PCDATA|to|from|header|message)>
```

### Restricciones de Ocurrencia

Las restricciones de ocurrencia indican la obligatoriedad de los elementos y el número de veces que un elemento puede aparecer en un documento, se expresan colocando un indicador de frecuencia en el elemento, por default los elementos son obligatorios y no repetibles, los indicadores de frecuencia para los elementos son los siguientes:

- "+" **1 o más (obligatorio, repetible).**
- "?" **0 o 1 (opcional, no repetible).**
- "\*" **0 o más (opcional, repetible).**

### Atributos

Los atributos pueden acompañar a un elemento para proveer información adicional sobre dicho elemento a través del valor que llevan asociado en la declaración, **atributo="valor"**. Dentro de la lista de atributos que puede llevar un elemento, se puede especificar su obligatoriedad, el tipo de valor que lleva, así como un posible valor por defecto.

La forma de definir la lista de atributos que puede tener un elemento es la siguiente:

Sintaxis:

```
<!ATTLIST nombre_elemento nombre_atributo tipo_atributo valor_default>
```

Ejemplo:

```
<!ATTLIST persona numero CDATA #REQUIRED>
```

XML válido:

```
<persona numero="5677" />
```

El valor para el parámetro **tipo\_atributo** puede ser uno de los siguientes:

Tipo	Descripción
<b>CDATA</b>	El valor es un dato carácter (Texto, datos no analizables por el analizador XML).
<b>(en1 en2 ..)</b>	El valor del atributo es uno de los declarados en la lista.
<b>ID</b>	Identificador que deberá ser único en el documento (no se puede especificar un valor por default).
<b>IDREF</b>	El valor del atributo es el valor id de otro atributo de algún elemento del documento.

El valor para el parámetro **valor\_default** de un atributo puede ser uno de los siguientes:

Valor	Descripción
<b>Valor</b>	El valor default de un atributo.
<b>#REQUIRED</b>	El atributo es obligatorio, es decir debe aparecer en el documento, y no puede tomar ningún valor por default.
<b>#IMPLIED</b>	El atributo no es requerido, es decir es opcional y puede no aparecer en el documento, en caso de aparecer no puede tomar ningún valor por default.
<b>#FIXED value</b>	El valor del atributo es fijo, por lo que no puede ser cambiado por el autor del documento.

## Entidades

Algunos caracteres tienen un significado especial en XML, así como el símbolo "<" que define el comienzo de una etiqueta, las entidades son expandidas cuando un documento es analizado por un analizador XML. Las siguientes entidades se encuentran predefinidas en XML.

Entidad	Carácter
<b>&amp;lt;</b>	<
<b>&amp;gt;</b>	>
<b>&amp;amp;</b>	&
<b>&amp;quot;</b>	"
<b>&amp;apos;</b>	'

## PCDATA (Parsed Character Data)

Texto que será fundamentalmente del documento, es decir, información que pueda ser analizada por un procesador de XML en busca de marcas especiales.

## CDATA (Character Data)

Son datos que aparecen principalmente como valores de atributos, el procesador de XML no busca ningún tipo de marca especial en ellos.

En la Figura 1.7 se muestra un ejemplo de una DTD que trata de libros que están en venta en una librería. La etiqueta `<!DOCTYPE>` especifica que debe haber al menos un elemento llamado libro, la siguiente etiqueta `<!ELEMENT>`, indica que libro contiene cuatro elementos: autor+, nombre+,

precio+ y editorial+, los cuales son obligatorios, posteriormente se realiza la declaración de todos los elementos de tipo #PCDATA, y por último la etiqueta <ATTLIST> indica que el elemento precio contiene un atributo llamado tipo, el cual es obligatorio y de tipo #PCDATA.

```
<?xml version="1.0"?>
<!DOCTYPE libro [
    <!ELEMENT libro (autor+, nombre+ ,precio+, editorial+)>
    <!ELEMENT autor (#PCDATA)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT precio (#PCDATA)>
    <!ATTLIST precio tipo PCDATA #REQUIRED>
    <!ELEMENT editorial (#PCDATA)>
]>
```

**Figura 1.7** Ejemplo de DTD

Algunas consideraciones a tomar en cuenta a la hora de decidir utilizar una DTD son las siguientes:

- No siempre es necesario validar un documento XML.
- No se puede declarar el tipo de cada elemento y de cada atributo, es decir, no se puede restringir que un elemento sea de tipo numérico y positivo. La falta de tal restricción es problemática para las aplicaciones de procesamiento e intercambio de datos, las cuales deben contener el código para verificar los tipos de los elementos y atributos.
- Es difícil usar un mecanismo DTD para especificar conjuntos desordenados de subelementos.
- Hay una falta de tipos en ID. Por ello no hay forma de especificar el tipo de elemento al cual se debería referir un atributo IDREF.
- No utiliza el lenguaje propio de XML, pero su implementación es más sencilla que la de un Schema XML.

En XML no existen reglas acerca de cuándo utilizar elementos o atributos, pero en XML es recomendable la utilización de elementos; algunos puntos considerados para llegar a esta conclusión son los siguientes:

- Los atributos no pueden tener múltiples valores.
- Los atributos no pueden ser expandidos fácilmente, en caso de futuros cambios.

- Los atributos no pueden describir estructuras debido a que no pueden tener otros valores.
- Los atributos son más difíciles de manejar con código de programación.
- Si se utiliza a los atributos como contenedores de datos, se pueden generar documentos difíciles de leer y mantener
- El propósito de los atributos es proveer información extra que no es muy relevante acerca de los elementos.
- Las asignaciones de referencias ID a los elementos se realizan mediante atributos, en este punto si se desea realizar referencias deben usarse atributos.

## 1.6 XML Schema

Para resolver las necesidades inherentes a XML que la DTD no satisface la organización W3C creó un lenguaje de esquema, XML Schema, que al igual que la DTD describe el contenido y la estructura de la información de los documentos que estén asignados al esquema, pero de una forma más precisa, los esquemas definen varios tipos predefinidos de datos como string, integer, decimal, date y boolean. Además permite tipos definidos por el usuario, que pueden ser tipos más simples con restricciones añadidas, o tipos complejos construidos con constructores como complexType o sequence. La Figura 1.8 muestra un ejemplo de un XML Schema que trata acerca de información manejada un banco: la información de los clientes, los impositores y las cuentas que manejan.

```

<xsd:schemaxmlns:xsd = «http://www.w3.org/2001/XMLSchema»>
<xsd:element name = «banco» type = «TipoBanco» />
<xsd:element name = «cuenta»>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = «numero-cuenta» type = «xsd:string»/>
      <xsd:element name = «nombre-sucursal» type = «xsd:string»/>
      <xsd:element name = «saldo» type = «xsd:decimal»/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name = «cliente»>
  <xsd:element name = «número-cliente» type = «xsd:string»/>
  <xsd:element name = «calle-cliente» type = «xsd:string»/>
  <xsd:element name = «ciudad-cliente» type = «xsd:string»/>
</xsd:element>

<xsd:element name = «impositor»>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = «nombre-cliente» type = «xsd:string»/>
      <xsd:element name = «número-cuenta» type = «xsd:string»/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name = «TipoBanco»>
  <xsd:sequence>
    <xsd:element ref = «cuenta» minOccurs = «0» maxOccurs = «unbounded»/>
    <xsd:element ref = «cliente» minOccurs = «0» maxOccurs = «unbounded»/>
    <xsd:element ref = «impositor» minOccurs = «0» maxOccurs = «unbounded»/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

**Figura 1.8** Ejemplo de XML Schema

Las definiciones de XML Schema se especifican en sintaxis XML. Para evitar conflictos con las etiquetas definidas por el usuario, se antepone a la etiqueta XML Schema la etiqueta del espacio de nombres “xs:” de XML Schema.

Explicando el ejemplo de la figura 1.8 el primer elemento es el elemento raíz (banco), cuyo tipo es TipoBanco, que se declara posteriormente; se definen los tipos de elementos: cuenta, cliente e impositor, el tipo cuenta es compuesto y se precisa como una secuencia de elementos: número\_cuenta, nombre\_sucursal y saldo. Cualquier tipo que tenga atributos o subelementos anidados se puede especificar como tipo complejo.

Alternativamente se puede especificar el tipo de un elemento como predefinido con el atributo type; obsérvese el uso de los tipos de XML Schema: xs:string y xs:decimal para restringir los tipos de los elementos de datos tales como numero\_cuenta.



El ejemplo define el tipo TipoBanco para contener cero o más apariciones tanto de cuenta como de cliente e impositor. Obsérvese también el uso de ref empleado para especificar la aparición de un elemento definido anteriormente; se puede definir el número mínimo y máximo de ocurrencias de subelementos mediante las palabras reservadas: minOccurs y maxOccurs. El valor predeterminado para ambos casos es uno, por lo que se tiene que especificar explícitamente para permitir cero o más apariciones.

Los atributos se especifican usando la etiqueta attribute. Por ejemplo, se podría haber definido el atributo numero\_cuenta, añadiendo la definición dentro de la declaración del elemento cuenta.

```
<xs:attributename= "numero_cuenta"/>
```

Al añadir el atributo use="required" a la especificación anterior, se declara que el atributo es obligatorio, mientras que el valor default de "use" es "optional", lo que indica que el atributo puede no aparecer. Las especificaciones de atributos pueden aparecer directamente dentro de la especificación complexType, aunque los elementos se aniden dentro de una especificación de secuencia. Se puede utilizar complexType para crear tipos complejos, la sintaxis es la misma que en el ejemplo, salvo que se añade el atributo name="nombreTipo" al elemento complexType, donde nombreTipo es el nombre que se desea para el tipo.

Además de definir tipos, XML Schema permite la especificación de claves y referencias a claves. En SQL una restricción de clave primaria o de unicidad asegura que los valores de un atributo no se dupliquen en la relación. En el contexto de XML, es necesario definir un ámbito en el que los valores sean únicos y formen una clave. Selector es una expresión de ruta que define el ámbito de la restricción, y las declaraciones field especifican los elementos o atributos que forman la clave. Para especificar que los números de cuenta forman una clave de los elementos cuenta en el elemento raíz banco, se podría añadir la siguiente restricción a la definición del esquema:

```
<xs:keyname= "claveCuenta">  
  <xs:selectorxpath= "/banco/cuenta"/>  
  <xs:fieldxpath= "numero_cuenta"/>  
</xs:key"/>
```

Se puede definir también la restricción de la clave externa desde impositor hasta cuenta:

```
<xs:keyrefname="claveExtImpositorCuenta" refer="claveCuenta" >
```

```
<xs:selectorxpath= "/banco/impositor"/>
  <xs:fieldxpath= "numero_cuenta"/>
<xs:key"/>
```

Obsérvese que el atributo refer indica el nombre de la clave a la que se hace referencia, mientras que field identifica los atributos que hacen la referencia.

Entre los beneficios que ofrece XML Schema respecto a una DTD se encuentran los siguientes:

- Permite crear tipos definidos por el usuario.
- Permite que el texto que aparece en los elementos esté restringido a tipos específicos, tales como, tipos numéricos en formatos específicos o incluso tipos más complicados como secuencias de elementos de otros tipos.
- Permite restringir los tipos para crear tipos especializados, por ejemplo, especificando valores mínimo y máximo.
- Permite la extensión de tipos complejos mediante el uso de una forma de herencia.
- Permite restricciones de unicidad y de clave externa.
- Está integrado con espacios de nombres para permitir a diferentes partes de un documento adaptarse a un esquema diferente.
- Se especifica mediante sintaxis XML.

A pesar de sus características, el precio a pagar por estas características es que XML Schema es significativamente más complicado que las DTDs.

## 1.7 Tipos de Datos (Estructurados, No Estructurados y Semi-estructurados)

Una clasificación de los datos de acuerdo a su estructura es: estructurados, no estructurados y semi-estructurados. Los cuales se explican brevemente a continuación (BLANCO, 2007).

**Datos Estructurados:** Este tipo de datos está caracterizado por:

- Estar organizados de alguna manera.
- Son atributos o variables con tipos definidos (int, float, string).
- Cada atributo en una relación está definido para todas las instancias de datos.
- Ejemplos: registros, base de datos relacional.

En la Figura 1.9 se muestra un ejemplo de una tabla que contiene datos estructurados, especificando el tipo de dato.

Nombre [char(10)]	cumpleaños [date]	sueldo [int]
Carlos	13-08-1980	5000
Juan	23-02-1977	7500

**Figura 1.9** Ejemplo de Datos Estructurados

**Datos No Estructurados:** Este tipo de datos tiene las siguientes características:

- No están organizados de acuerdo a algún patrón.
- No poseen definiciones de tipos.
- No existe el concepto de variables o atributos.
- Ejemplo: documentos de texto sin estructura.

En la Figura 1.10 se muestra un ejemplo de un texto que contiene datos no estructurados.

“Carlos nació el 13 de Agosto de 1980. El tiene un sueldo de 5000. Alguien más nació el 23 de Febrero de 1977, su nombre es Juan y su salario es de 7500”

**Figura 1.10** Ejemplo de Datos No Estructurados

**Datos Semi-estructurados:** Este tipo de datos tiene las siguientes características:

- Pueden ser irregulares y no respetar un esquema en particular.
- La estructura puede evolucionar muy rápidamente, ya que se pueden agregar características con el tiempo fácilmente.
- Los nuevos datos pueden no respetar la estructura de los datos existentes previamente.
- Alta frecuencia de modificaciones en las propiedades estructurales.
- Datos débilmente tipados.
- Son descritos por sí mismos (no existe una separación entre la descripción del tipo, estructura y valor).
- Pueden no tener esquema o tener un esquema que impone restricciones débiles sobre los datos.

- Ejemplo las variables que probablemente pertenecen a un tipo de datos u otro (x=1 es válido y x="hola" también es válido).
- Una instancia de datos que no necesariamente tiene todos los atributos definidos. En un ambiente de datos semi-estructurados se pueden omitir atributos en algunas instancias de datos lo que permite tener instancias de datos heterogéneas.
- Ejemplos: documentos XML ya que son un estándar para la publicación e intercambio de datos en el Web donde los datos son semi-estructurados.

## 1.8 Tipos de Documentos XML

Los documentos XML pueden ser clasificados en base a los datos que contienen: **centrados en datos y centrados en documentos**.

Los centrados en datos son documentos para el intercambio de datos, contienen datos actualizables usados en maneras diversas y los distintos datos que se transmiten, son partículas atómicas bien definidas. Este tipo de documentos capturan datos estructurados como los que se encuentran en un catálogo de productos, una orden de compra o una factura por ejemplo.

Los centrados en el documento tienen una estructura irregular, aunque posean formato no es tan estricto y definido, decimos en este caso que son semi-estructurados, tienden a ser más impredecibles en tamaño y contenido que los centrados en los datos, con tipos de datos de tamaño limitado y reglas menos flexibles para campos opcionales y contenido. Este tipo de documentos por el contrario capturan datos no estructurados como artículos, libros o e-mails por ejemplo. Las Figura 1.11 proporciona un ejemplo de un documento XML centrado en documento.

```

<nota>
  <para>Juan</para>
  <de>Emanuel</de>
  <encabezado>Recordatorio</encabezado>
  <cuerpo>No olvides la comida del fin de
  semana</cuerpo>
</nota>

```

**Figura 1.11** Documento XML Centrado en Documentos

Los sistemas de almacenamiento XML deben acomodarse eficientemente con ambos tipos de datos, dado que, XML está siendo ampliamente usado en sistemas que administran ambos tipos de datos. La mayoría de los productos se enfocan en servir uno de esos formatos de datos mejor que el otro.

Las bases de datos relacionales son típicamente mejores al tratar con requerimientos centrados en los datos, mientras que los sistemas de administración de contenido y de documentos son típicamente mejores para almacenar datos centrados en el documento.

## Capítulo 2. Bases de Datos XML

---

### 2.1 La Importancia de las Bases de Datos

El almacenamiento de datos es uno de los procesos que se halla en muchos de los sistemas de información. Una base de datos es una colección de datos interrelacionados entre sí, que contiene la información que la organización que se desea administrar.

Un sistema gestor de bases de datos (SGBD) consiste en la base de datos misma y un conjunto de programas para acceder a los datos. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos, de manera que sea tanto práctica como eficiente. Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar información como la provisión de mecanismos para la manipulación de la misma. Además, los sistemas de bases de datos deben proporcionar fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

Las bases de datos son ampliamente utilizadas, forman una parte esencial en la mayoría de las empresas actuales. A lo largo de las últimas cuatro décadas del siglo veinte, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaron directamente con los sistemas de bases de datos, aunque sin darse cuenta hicieron uso de estos (con los informes impresos como extractos de tarjetas de crédito, o mediante agentes como cajeros de bancos y agentes de reserva de líneas aéreas); después vinieron los cajeros automáticos. La revolución de Internet a finales de la década de 1990 aumentó significativamente el acceso directo del usuario a las bases de datos. Las organizaciones convirtieron muchas de sus interfaces en interfaces Web, y pusieron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una tienda de libros en línea y se busca un libro o una colección de música se está accediendo a datos almacenados en una base de datos. Así, aunque las interfaces de datos ocultan detalles del acceso a las bases de datos, y la mayoría de la gente ni siquiera es consciente de esto, el acceso a bases de datos forma una parte esencial de la vida de casi todas las personas actualmente.

## 2.2 XML y Bases de Datos

XML se ha convertido un lenguaje ampliamente utilizado para la representación de datos semi-estructurados, siendo actualmente el estándar para el intercambio de información e integración de datos y de aplicaciones(BARRIOS, 2009).

XML actualmente es utilizado en la gestión documental: en la edición y publicación electrónica, en la gestión de grandes volúmenes de documentos, en el intercambio de información entre sistemas heterogéneos, en la documentación técnica de manuales y en buscadores inteligentes debido a que la información en los documentos XML está etiquetada por su significado de forma precisa.

Es importante destacar que XML aporta mucha potencia y flexibilidad a las aplicaciones basadas en la Web, proporcionando numerosas ventajas a los programadores y usuarios como lo son: búsquedas con más significado, programación de aplicaciones *WEB* flexibles, integración de datos procedentes de fuentes dispares y la capacidad de buscar en varias bases de datos no compatibles entre sí(HERNÁNDEZ, 2005).

XML es un estándar potente y de amplia aceptación para guardar y comunicar información acerca de objetos; esto es: documentos, imágenes, fotos etc., permite la codificación de información separada de la forma en la que se debería presentar al usuario.

En esencia los datos XML son simplemente datos Unicode o ASCII que se pueden ver como un documento de texto estructurado lógicamente. Una base de datos XML se puede ver como una colección de documentos XML en la que cada documento representa un registro. En los últimos años se ha visto necesaria la existencia de estándares de intercambio de información, con el objetivo de que las organizaciones puedan compartir su información de una manera más cómoda, automática y eficiente. En lo que concierne a las bases de datos, XML permite integrar los siguientes sistemas de información hasta ahora separados:

- Sistemas de información basados en documentos, los cuales tienen estructura irregular, anidada, utilizan tipos de datos relativamente simples y dan gran importancia al orden.
- Sistemas de información estructurados los cuales tienen una estructura muy regular, son relativamente planos, utilizan tipos de datos complejos y dan poca importancia al orden.

El hecho de que el XML permita expresar información de características y estructuras muy diversas, no quiere decir que XML venga a sustituir a las bases de datos tradicionales. XML no es una base de datos ni un modelo de datos, es simplemente un lenguaje de marcas y un documento de texto.

Cuando se entiende a un documento XML y a su DTD asociada, se puede ver la comparación de esta tecnología con las bases de datos, ya que aparecen muchas cosas en común, sin que éste sea una base de datos:

- XML utiliza uno o más documentos para almacenar la información.
- Define esquemas sobre la información (DTD's y XML Schema).
- Tiene lenguajes de consulta específicos para recuperar la información requerida.

XML se está convirtiendo en el formato de datos de elección para una amplia variedad de soluciones de sistemas de información. Los beneficios que a menudo se asocian con la utilización de XML incluyen independencia de la plataforma, bajo costo de entrada, y la habilidad de compartir datos de manera transparente. XML puede además acomodar una variedad de datos incluyendo texto, imágenes y sonido. Sin embargo, nuevos requerimientos de habilidades y problemas administrativos causados por la creciente cantidad de datos XML deben también ser administrados. La mayoría de las aplicaciones tradicionales de negocio y de las aplicaciones basadas en Internet dependen de bases de datos. Existen diversos modelos diferentes de bases de datos, sin embargo, la mayoría son relacionales. Para mantener datos en una base de datos, éstos se deben obtener y almacenar de una manera consistente, confiable y eficiente.

En la actualidad, existen diferentes soluciones para el almacenamiento de documentos XML, las cuales se pueden clasificar en dos conjuntos: las bases de datos nativas y extensiones XML a las bases de datos, que permiten el almacenamiento de documentos XML en SGBD convencionales, normalmente relacionales.

### 2.3 Bases de Datos Habilitadas para XML

Debido al incremento de la información almacenada en formato XML, surge la necesidad de guardar estos documentos en bases de datos.



Actualmente los Sistemas Gestores de Bases de Datos Relacionales (SGBD) cuentan con capacidad para el almacenamiento, tratamiento y obtención de documentos XML, entre los principales se puede mencionar: Microsoft SQL Server, Oracle y Sybase ASE; el inconveniente es que el almacenamiento de estos documentos es en formato relacional, es decir, guardan los documentos en forma tabular haciendo una transformación o en el mejor de los casos guardan el documento completo en formato Binary Large Object (BLOB); una de las principales características de estos SGBD es que muestran los resultados de las consultas en formato XML, por lo que son conocidos como SGBD XML-enabled.

En documentos centrados en datos se puede realizar un mapeo entre los distintos elementos definidos en el documento y el modelo de datos del SGBD, esta posibilidad es viable, debido a que los datos poseen una estructura regular y controlada siendo fácil de transformar en un esquema relacional, sin embargo, posee el inconveniente que sólo se almacenan los datos que nos interesan conservar, y partes del documento son perdidas durante la transformación, y a la hora de reconstruir el documento a partir de los datos almacenados existe la posibilidad de obtener un documento distinto.

La opción de almacenar el documento entero sobre un campo BLOB en una tabla de la base de datos tiene como ventaja que mantiene el formato del documento, pero el gran inconveniente es no poder realizar en principio ninguna operación de consulta sobre su contenido.

Se debe tener en cuenta que los grandes fabricantes de SGBD están aumentando las capacidades de tratamiento XML de sus productos, incluyendo características de SGBD XML nativos, por lo que en un futuro próximo es posible que la frontera entre estos dos tipos de gestores quede diluida.

## 2.4 Bases de Datos Nativas XML

La solución para el almacenamiento de datos en formato XML es la de disponer de sistemas de gestión de información que sean capaces de gestionar datos en XML directamente (HERNÁNDEZ, 2005).

Las bases de datos XML Nativas (NXD) surgen por la necesidad de una gestión eficiente de grandes cantidades de documentos XML, argumentando que los documentos XML no se pueden almacenar en SGBD convencionales debido a su naturaleza jerárquica y semi-estructurada(M.Thomas Connolly., 2005).

No existe una definición estándar para una base de datos nativa en XML, pero la organización XMLDB Initiative for XML Databases describe una base de datos de este tipo como: "Un modelo lógico para documentos XML el cual almacena y recupera documentos de acuerdo a dicho modelo"(CORNEJO, 2002).

Los productos de bases de datos XML nativas están centrados en el almacenamiento y gestión de documentos XML. Este tipo de gestores tienen las siguientes características:

- Definen un modelo de datos XML, donde especifica qué elementos son lógicamente significativos. Teniendo en cuenta a los elementos, atributos, texto y orden del documento.
- Utilizan el documento XML como unidad mínima de almacenamiento.
- Pueden utilizar cualquier estrategia de almacenamiento.
- Los documentos son almacenados y recuperados de acuerdo al modelo de datos definido.
- No necesitan un modelo de almacenamiento físico en particular.

La principal ventaja de la utilización de un SGBD es el manejo de los documentos centrados en contenido, debido a que se pueden almacenar directamente sin tener que aplicar algoritmos de transformación y mapeo. Los SGBD XML nativos implementan algoritmos de búsqueda e índices específicos que aceleran el acceso a los documentos.

La mayoría de las bases de datos XML usan un modelo lógico para agrupar documentos llamados colecciones. Actualmente las NXD soportan al menos un lenguaje de consulta. Existen diversas NXD comerciales, las cuales brindan diferentes características, sin embargo, la mayoría de ellas ofrece: procesamiento de datos, almacenamiento y búsquedas.

## 2.5 Lenguajes de Consulta

Dado el creciente número de aplicaciones que usan XML para intercambiar, transmitir y almacenar datos, las herramientas para la gestión efectiva de datos XML están siendo cada vez más importantes. En particular las herramientas para consultar los datos XML son esenciales para extraer información y para convertir datos entre distintas representaciones (esquemas) en XML. Al igual que la salida de una consulta relacional es una relación, la salida de una consulta XML puede ser un documento XML. Como resultado, la consulta y la transformación se pueden combinar en una única herramienta(SILBERSCHATZ, 2006).

### 2.5.1 XPath

XPATH<sup>6</sup> es un lenguaje de consulta desarrollado por el equipo de trabajo del W3C<sup>7</sup>, el cual construye expresiones que recorren y procesan un documento XML utilizando la estructura jerárquica del documento, es definido como un lenguaje para obtener partes del documento XML. Los lenguajes de consulta XPath y XQuery son utilizados en bases de datos nativas con XML.

La forma en que XPath manipula los datos es a través de un parser, el cual construye un árbol de nodos. El árbol presenta las características de cualquier árbol de datos, es decir, tiene raíz, hijos, ancestros, descendientes, etc.(MENDOZA, 2007).

XPath realiza el análisis de documentos XML aplicando una expresión al texto de un documento XML, realiza la navegación del documento a través de elementos y atributos; los valores y elementos que coinciden con la expresión construyen el resultado de la consulta(POWELL, 2007).

Para procesar un documento XML, XPath toma en cuenta que dicho documento tiene estructura jerárquica por lo que inicialmente un analizador lo recorre y forma un árbol de nodos. El árbol contiene un nodo raíz y desde él cuelgan otra serie de nodos que se extienden hacia abajo, hasta llegar a los nodos hoja, los cuales pueden contener sólo texto, comentarios, instrucciones de proceso o incluso estar vacíos y sólo contener atributos. Un caso especial de nodos son los que pertenecen a los atributos, debido a que un elemento puede contener el número de atributos que

---

<sup>6</sup> Acrónimo de XML Path Language

<sup>7</sup> Acrónimo de World Wide Web Consortium

desea, así que por cada uno de éstos se le creará un nodo atributo, aunque estos no son considerados como hijos sino como etiquetas del nodo elemento(AGUILAR, 2010).

Un nodo es una representación lógica de una parte de un documento XML. En XPath existen 7 tipos de nodos(SOLTILLO, 2010):

- **Nodo Raíz:** Es el documento en sí mismo.
- **Nodo Elemento:** Cada parte del documento está representado por un nodo Elemento.
- **Nodo Atributo:** Cada atributo del documento está representado por un nodo atributo. El elemento al que está asociado dicho atributo es su **Nodo Padre**.
- **Nodo Texto:** Representa el contenido textual de un elemento, evidentemente tiene valor de cadena pero no nombre.
- **Nodo Comentario:** Representa un comentario.
- **Nodo Instrucción de Procesamiento:** Representa una instrucción de procesamiento dentro del documento.

XPath trata las partes de un documento XML mediante expresiones de rutas de acceso. Una expresión de ruta en XPath es una secuencia de pasos de ubicación separados por «/». El resultado de la expresión de ruta es un conjunto de valores(SILBERSCHATZ, 2006).

```
<banco-2>
  <cuota numero-cuenta = «C-401» tenedores = «C100 C102»>
    <nombre-sucursal> Centro </nombre-sucursal>
    <saldo> 500 </saldo>
  </cuota>
  <cuota numero-cuenta = «C-402» tenedores = «C102 C101»>
    <nombre-sucursal> Navacerrada </nombre-sucursal>
    <saldo> 900 </saldo>
  </cuota>
  <cliente cliente-id = «C100» cuentas = «C-401»>
    <nombre-cliente>Juncal</nombre-cliente>
    <calle-cliente> Mártires </calle-cliente>
    <ciudad-cliente> Melilla </ciudad-cliente>
  </cliente>
  <cliente cliente-id = «C101» cuentas = «C-402»>
    <nombre-cliente>Loreto</nombre-cliente>
    <calle-cliente> Montaña </calle-cliente>
    <ciudad-cliente> Cáceres </ciudad-cliente>
  </cliente>
</banco-2>
```

**Figura 2.1** Ejemplo de Documento XML

Un ejemplo de consulta XPath basada en el documento de la Figura 2.1, es conocer a todos los clientes del banco; la expresión Xpath para la consulta es la siguiente: **/banco-2/cliente/name**

Una vez analizado el documento el resultado de la consulta que devolverá el siguiente resultado:

**<name>Juncal</name>**

**<name>Loreto</name>**

En la expresión utilizada para la consulta el símbolo “/” inicial indica la raíz del documento (nótese que esto es una raíz abstracta <banco-2>, que es el nodo raíz del documento). Las expresiones de ruta se evalúan de izquierda a derecha. Cuando se evalúa la expresión de ruta el resultado de la ruta en cualquier punto consiste en un conjunto de nodos del documento. Cuando el nombre del elemento cliente aparece antes de la siguiente ‘/’, se refiere a todos los elementos del nombre especificado que son hijos de elementos en el conjunto de elementos actual. Puesto que varios hijos pueden tener el mismo nombre, el número de nodos en el conjunto de nodos puede aumentar o decrecer con cada paso. También se puede acceder a los valores de atributo mediante el uso del símbolo «@». Por ejemplo, **/banco-2/cuenta/@numero-cuenta**, devuelve un conjunto con todos los valores de los atributos número-cuenta de los elementos cuenta. XPath tiene las siguientes características:

- La selección de predicados puede seguir cualquier paso en una ruta y están contenidos entre corchetes. Por ejemplo, **banco-2/cuenta[saldo > 400]**, devuelve los elementos cuenta con un valor saldo mayor que 400, mientras que, **/banco-2/cuenta[saldo > 400]/@numero-cuenta**, devuelve los números de cuenta de dichas cuentas. Se puede comprobar la existencia de un subelemento mediante su listado sin ninguna operación de comparación; por ejemplo si se elimina «>400» de la expresión anterior devolvería los números de cuenta de todas las cuentas que tiene un subelemento saldo, sin considerar su valor.
- XPath proporciona varias funciones que se pueden usar como parte de predicados incluyendo la comprobación de la posición del nodo actual en el orden de los nodos hermanos y contando el número de nodos coincidentes. Por ejemplo, la expresión de ruta **banco-2/cuenta/[cliente/count() > 2]**, devuelve las cuentas con más de dos clientes. Se pueden usar las conectivas lógicas AND y OR en los predicados y la función NOT.

- La función `id(«foo»)` devuelve el nodo (si existe) con un atributo del tipo ID y cuyo valor sea «foo». La función `id` se puede incluso aplicar a conjuntos de referencias o incluso a cadenas que contengan referencias múltiples separadas por espacios vacíos, tales como IDREFS. Por ejemplo, la ruta, `/banco-2/cuenta/id(@tenedores)`, devuelve la lista de todos los clientes referenciados desde el atributo `tenedores` de los elementos `cuenta`.
- El operador “|” permite unir resultados de expresiones. Por ejemplo, si la DTD de `banco-2` también contiene elementos para préstamos con atributo `prestamista` del tipo IDREFS para identificar el prestamista de un préstamo, la expresión `/banco-2/cuenta/id(@tenedores)|/banco-2/préstamo/id(@prestamista)`, proporciona la lista de clientes con cuentas o préstamos. Sin embargo, el operador “|” no se puede anidar dentro de otros operadores.
- Una expresión XPath puede saltar varios niveles de nodos mediante el uso de `«//»`. Por ejemplo, la expresión `/banco-2//name` encuentra cualquier elemento `name` en cualquier lugar bajo el elemento `/banco-2`, sin considerar al elemento en el que está contenido. Este ejemplo ilustra la capacidad de buscar datos requeridos sin un conocimiento completo del esquema.

### 2.5.2 XQuery

De acuerdo al incremento de información que es almacenada, intercambiada y presentada usando XML, la habilidad para consultar fuentes de datos XML llega a ser muy importante. XML tiene la flexibilidad de representar diferentes tipos de información desde diversas fuentes. Para explotar esta flexibilidad, un lenguaje de consulta XML debe proveer componentes para obtener e interpretar la información desde diversas fuentes. Como resultado de esta necesidad surge XQuery (AGUILAR, 2010).

XQuery fue desarrollado por el W3C como lenguaje de consulta normalizado para XML y forma parte de sus recomendaciones desde el 2007. XQuery procede de un lenguaje de consulta XML denominado Quilt, el cual incluía características de lenguajes anteriores tales como XPath y otros dos lenguajes de consultas XML: XQL y XML-QL (SILBERSCHATZ, 2006).

XQuery extrae y manipula toda aquella información que se encuentra en formato XML, para ejecutar una consulta, la expresión dentro del cuerpo de ésta se evalúa y regresa el resultado

obtenido. Este lenguaje utiliza expresiones XPath para acceder a las partes del documento donde se encuentra la información que se desea extraer.

Las consultas XQuery se basan en SQL, pero difieren significativamente; la estructura de las consultas es conocida como “FLWOR”, ya que denotan las 5 secciones en las que se organizan: **for**, **let**, **orderby** y **return**. Por ejemplo, la siguiente expresión aplicada al documento de la Figura 2.1 devuelve los números de cuenta de las cuentas corrientes.

```
for $x in /banco-2/cuenta
let $numcuenta : = $x/@numero-cuenta
where $x/saldo > 400
return<numero-cuenta> $numcuenta</numero-cuenta>
```

La cláusula **for** es como la cláusula from de SQL y proporciona una serie de variables cuyos valores son los resultados de expresiones XPath; la cláusula **let** simplemente permite que se asigne el resultado de las expresiones XPath al nombre de las variables, esto por simplicidad de representación; La cláusula **where**, ejecuta comprobaciones adicionales sobre los resultados encontrados en la cláusula for; la cláusula **orderby** permite la ordenación de la salida. Finalmente la cláusula **return** permite la construcción de resultados XML.

Una consulta no necesita tener todas las secciones, puede contener solamente las cláusulas for y return. Sin embargo la cláusula let simplifica las consultas complejas. En la cláusula return respecto al uso de llaves “{}”, cuando XQuery encuentra un elemento como <numero-cuenta> que inicia una expresión, trata su contenido como texto normal, excepto las partes encerradas entre llaves, que se evalúan como expresiones. Los contenidos dentro de las llaves son, no obstante, tratados como expresiones a evaluar.

XQuery proporciona una forma de construir elementos, usando los constructores **element** y **attribut**, por ejemplo, si la cláusula return de la consulta anterior se reemplaza por la siguiente cláusula, la consulta devolvería elementos cuenta con numero\_cuenta y nombre\_sucursal como atributos y saldo como subelemento.

```
return element cuenta {
  attribute numero_cuenta{$x/@numero_cuenta},
  attribute nombre_sucursal{$x/@nombre_sucursal},
  element saldo{$x/saldo}
}
```

Las expresiones de ruta XQuery son las mismas expresiones que en XPath, pueden devolver un único valor, un elemento o una secuencia de ellos. XQuery tiene una definición interesante de las operaciones de comparación sobre secuencias. Por ejemplo, la expresión **\$x/saldo>400** tendría la interpretación usual si el resultado de **\$x/saldo** fuese un único valor, pero si el resultado es una secuencia de que contiene varios valores, la expresión se evalúa a verdadero si al menos uno de los valores es mayor a 400.

En el siguiente ejemplo de consulta se observa la utilización de las 5 cláusulas, esta consulta extrae la información con el XPath **//libro** del documento libros.xml, obteniendo con la cláusula **let** los autores y asignándolos a la variable **\$c**, la condición **where** filtra los libros con más de de 5 autores y **orderby** ordena las el resultado por título, devolviendo como resultado únicamente los títulos.

```
for $libro1 in doc("libros.xml")//libro
let $c:= libro1//autor
where count($c) > 5
orderby $libro1/titulo
return $libro1/titulo
```

## 2.6 Sistemas Gestores de Bases de Datos XML Nativos

Las bases de datos XML Nativas son bases de datos que almacenan XML usando un formato que permite un procesamiento más rápido. Un SGBD habilitado para XML sólo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos, mientras que un SGBD XML nativo debe manejar todos los tipos de documentos posibles.

Una vez definido el concepto de SGBD XML se describen dos implementaciones distintas de estos, una comercial "Tamino" y otra open source "eXist".

### 2.6.1 Tamino

Tamino es el SGBD nativo de la empresa SoftwareAG, es un producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles. La arquitectura de Tamino tiene los siguientes componentes básicos:



- **Native XML Data Store + XML Engine:** Es el componente central de la arquitectura, contiene el parser XML, el almacén nativo de datos y el intérprete de consultas.
- **Data Map:** Es el almacén de metadatos, contiene información acerca de: cómo validar esquemas, almacenar e indexar información y mapeo de estructuras.
- **X-Node:** Es el componente para dar acceso a base de datos externas, mapeando los datos a estructuras XML. El acceso a esta información es transparente para el usuario y se accede a ella cada vez que se necesita, es decir, que no es replicada.
- **X-Tension:** Permite la definición de funciones de usuario para ampliar las prestadas por Tamino.
- **Tamino Manager:** Herramienta gráfica de gestión.

Los documentos se almacenan en una base de datos propia y no se transforman en otro modelo. Existe un espacio separado para documentos y otro para índices. Un doctype es el elemento raíz de un DTD o XML Schema, es decir, el elemento que define el comienzo y el final del documento.

La base de datos está estructurada en colecciones, una colección es un conjunto de documentos, de modo que es una estructura de árbol donde cada documento pertenece a una única colección. Cada colección tiene asociado varios doctypes. El elemento raíz del documento XML define a que doctype estará asociado el documento, en caso de no poseer ninguno, éste se crea dinámicamente. Esto posibilita el almacenamiento de documentos sin formato definido.

La colección también tiene asociado un Schema con información tanto física como lógica de la colección. La parte lógica define las relaciones y propiedades de los documentos XML y la física contiene información sobre el almacenamiento e indexación de los mismos.

También permite el almacenamiento de documentos no-XML, para estos existe un doctype especial llamado nonXML. El gestor asigna a cada documento un identificador, y el usuario puede asignarle un nombre, el cual debe de ser único dentro de cada doctype, y puede ser usado para acceder directamente al fichero a través de su URL. Los elementos de configuración del sistema también son documentos XML almacenados en la colección system, por lo que pueden ser accedidos y manipulados por las herramientas estándar proporcionadas.

Tamino provee índices que son mantenidos automáticamente cuando los documentos son añadidos, borrados o modificados. Modificando el esquema de la colección se pueden definir distintos tipos de índices para optimizar consultas.

El lenguaje de consulta de datos utilizado es XQuery; Permite la modificación de documentos, a través de operaciones de inserción, borrado, reemplazo y renombrado, utilizando extensiones propias. Es un SGBD completo, con todas las funcionalidades que se pueden pedir a un SGBD moderno, como soporte para transacciones, multiusuario, log de operaciones, herramientas para cargas masivas y sistema completo de backups, lo cual hace de Tamino un producto escalable y con buen rendimiento.

### 2.6.2 eXist

eXist es un SGBD XML nativo open source que tiene las funcionalidades básicas de cualquier gestor nativo, además de integrar algunas técnicas avanzadas como: búsquedas de términos, búsquedas por proximidad de términos y búsquedas basadas en expresiones regulares. La arquitectura de eXist posee los siguientes componentes:

El motor de base de datos está completamente escrito en Java y posee distintos módulos para el almacenamiento de datos. En estos momentos el almacén principal está compuesto por una base de datos nativa, además de incluir la posibilidad de almacenamiento sobre base de datos relacionales. En principio esta arquitectura oculta completamente la implementación física del motor de la base de datos. Es destacable que el motor de base de datos es muy compacto y puede funcionar tanto en modo servidor, incrustado en una aplicación o en un contenedor J2EE (como Tomcat).

Como gestor XML nativo soporta los estándares de consulta XPath y XQuery, además de extensiones propias para la actualización. El SGBD incluye aplicaciones que permiten ejecutar consultas directamente sobre la Base de Datos.

Los documentos se almacenan en colecciones, las cuales pueden estar anidadas; desde un punto de vista práctico el almacén de datos funciona como un sistema de archivos. Cada documento está

en una colección. Los documentos no tienen que tener una DTD ó XML Schema asociado, y dentro de una colección pueden almacenarse documentos de cualquier tipo.

El almacén central nativo de datos es el archivo dom.dbx; es un archivo paginado, donde se almacenan todos los nodos del documento de acuerdo al modelo DOM del W3C. Dentro del mismo archivo existe también un árbol B+, que asocia el identificador único del nodo con su posición física. El archivo collections.dbx almacena la jerarquía de colecciones y relaciona a esta con los documentos que contiene; se asigna un identificador único a cada documento de la colección que es almacenado también junto al índice.

El gestor automáticamente indexa todos los documentos utilizando índices numéricos para identificar los nodos del mismo (elementos, atributos, texto y comentarios). Los índices están basados en árboles B+ y definidos a nivel de colección para mejorar el rendimiento. Durante la indexación se asigna un identificador único a cada documento de la colección que es almacenado también junto al índice.

El almacén para el índice de elementos y atributos está en el archivo elements.dbx. Para ahorrar espacio los nombres de los nodos no son utilizados para construir el índice, en su lugar se asocia el nombre del elemento y de los atributos con unos identificadores numéricos en una tabla de nombres, y cada entrada del índice consiste en entradas con clave < id colección, id nombre> y una relación de valores de document-id y node-id que le corresponden.

Por defecto eXist indexa todos los nodos de texto y valores de atributos, dividiendo el texto en palabras; en el fichero words.dbx se almacena esta información, en este caso cada entrada del índice está formada por un par <colección id, palabra> y después una lista al nodo que contiene dicha palabra. También pueden definirse índices específicos para acelerar consultas.

eXist posee funciones de backup a través del cliente Java ó de la consola Unix, cuando se hace el backup se exporta el contenido de la base de datos como documentos XML estándar y se crea una jerarquía de directorios a imagen de la jerarquía de colecciones. Se guarda información sobre configuración de índices, usuarios y un archivo especial llamado contents.xml con metadatos sobre el contenido (permisos, propietario, fechas de modificación, etc). Tiene también

funcionalidades de recuperación en caso de caída del sistema, deshaciendo transacciones incompletas y rehaciendo las terminadas pero no reflejadas.

## 2.7 Uso de Bases de Datos XML Nativas

El único requisito indispensable para una aplicación que desee utilizar una base de datos XML nativa, es que debe utilizar XML, más allá de esto, no hay reglas para determinar qué tipo de aplicaciones requieren la construcción de una base de datos XML nativa (HERNÁNDEZ, 2005).

Los entornos comerciales suelen tener cantidades incontrolables de los datos de forma masiva. Cuanto más una base de datos relacional se divide en elementos autónomos, las consultas pueden llegar a ser más complejas. El bajo rendimiento es inaceptable en la industria debido a que las empresas deben obtener ganancias para seguir siendo viables.

El resultado es que, comercialmente es más probable que XML se utilice con el fin de simplificar las operaciones de transferencia de datos. Hay muchos entornos comerciales que realizan el uso de XML, para manejar cuestiones complejas de datos. Las áreas más comunes del uso de bases de datos XML son las siguientes:

- Transferencia de datos y B2B (Business to Business): portales de información o mecanismos de transferencia de datos en el mundo corporativo.
- Catálogo y la gestión documental: Catálogos de datos, tales como, el almacenamiento de documentos de texto científicos y trabajos de investigación que requieren amplios mecanismos de búsqueda tanto como títulos de los documentos y contenido dentro de los mismos.
- Medicina: La información médica incluye a menudo diagnósticos especializados de datos multimedia, como imágenes, e incluso las anotaciones de imágenes. Una vez más, la complejidad es el problema para el uso de XML.
- Servicios web: En cualquier tipo de servicio Web que proporcione datos en tiempo real, se puede utilizar XML para estandarizar tanto la transferencia como la pantalla.

- La bioinformática y la genética: Este tipo de datos es complejo y muy volátil en particular. Volatilidad implica que estos campos de investigación pueden cambiar dramáticamente, y en repetidas ocasiones. La capacidad de flexibilidad y complejidad de manejo de XML se pueden beneficiar enormemente estos campos.
- Datos incompletos e inconsistentes: Cualquier información, incluyendo datos como el perfil del cliente y los datos de entretenimiento a menudo puede ser incompleta. XML es lo suficientemente flexible para que pueda dejar cosas fuera y mezclar las cosas de distinto tipo. Por ejemplo, el entretenimiento podría incluir datos sobre restaurantes, clubes nocturnos, hoteles, etc. XML permite este tipo de flexibilidad porque tanto sus datos y la estructura de metadatos son flexibles y están disponibles directamente. En otras palabras, los datos y metadatos se encuentran en el mismo lugar.

## Capítulo 3. Diseño de Bases de Datos

---

### 3.1 Importancia del Diseño de una Base de Datos

El diseño de las bases de datos es el proceso que ayuda a determinar la organización de una base de datos, incluidos: su estructura, contenido y las aplicaciones por desarrollar (BATINI, 1994). Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo se ha progresado mucho en el diseño de bases de datos y ésta se considera ahora una disciplina estable, con métodos y técnicas propias. Debido a la gran aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información de la mayoría de las organizaciones.

El diseño de bases de datos se ha convertido en una actividad popular, desarrollada no sólo por profesionales sino también por especialistas. En 1960 cuando las bases de datos entraron por primera vez en el mercado de software, los diseñadores actuaban como artesanos, con herramientas muy primitivas y el diseño se confundía muchas veces con la implantación.

La tarea de creación de aplicaciones de bases de datos es una labor compleja que implica varias fases como: el diseño del esquema, el diseño de los programas que tienen acceso a los datos para su actualización y el diseño del esquema de seguridad para controlar el acceso a los datos. Las necesidades de los usuarios desempeñan un papel central en el proceso de diseño.

El diseño de un entorno de aplicaciones de bases de datos que respondan a las necesidades de la empresa que se está modelando, exige prestar atención a un amplio conjunto de consideraciones. Estos aspectos adicionales del uso esperado de la base de datos influyen en gran variedad de opciones de diseño en los niveles físico, lógico y de vistas.

#### **Fases del diseño:**

En aplicaciones pequeñas puede resultar factible para un diseñador que comprenda los requisitos de la aplicación, decidir directamente las relaciones que hay que crear, sus atributos y las restricciones sobre las relaciones. Sin embargo, un proceso de diseño tan directo resulta difícil para las aplicaciones reales ya que a menudo son muy complejas (SILBERSCHATZ, 2006).

Frecuentemente no existe una sola persona que comprenda todas las necesidades de datos de la aplicación. El diseñador debe interactuar con los usuarios para comprender las necesidades de la aplicación, realizar una aplicación de alto nivel de esas necesidades que pueda ser comprendida por los usuarios y luego traducir esos requisitos a niveles inferiores de diseño. Los modelos de alto nivel ofrecen a los diseñadores un marco conceptual, en el que pueden especificar de forma sistemática los requisitos de datos de los usuarios y la estructura de la base de datos que satisface esos requisitos. La fase inicial del diseño de una base de datos es la caracterización completa de las necesidades de datos de los posibles usuarios. El resultado de esta fase es una especificación de requisitos del usuario.

Debido a que el diseño de bases de datos está fuertemente influido por la elección de modelos adecuados para representar los datos y las funciones, el diseñador elige el modelo de datos y aplicando los conceptos del modelo de datos elegido, traduce esos requisitos en un esquema conceptual de la base de datos. Los modelos poseen un conjunto fijo de construcciones lingüísticas que pueden ser usados en la descripción de datos. Es importante hacer notar que las construcciones de un modelo también cuentan con una representación gráfica, la cual permite al diseñador crear diagramas y dibujos. Estos documentos deben ser fáciles de leer y entender, ya que son ingredientes esenciales en el proceso de diseño.

El modelo conceptual no se ayuda mucho de herramientas automáticas, el diseñador asume total responsabilidad sobre el proceso de entender y transformar los requerimientos en esquemas conceptuales, así pues se cree que el diseño conceptual es la fase crucial del diseño de la base de datos.

El esquema desarrollado en la fase de diseño conceptual proporciona una visión detallada de la empresa. En términos de este modelo, el esquema conceptual especifica las entidades que se representan en la base de datos, sus atributos, las relaciones entre ellas y las restricciones que la afectan. El diseñador revisa el esquema para confirmar que realmente se satisfagan los requisitos y que no entran en conflicto entre sí y revisa el diseño para eliminar características redundantes; su atención se centra en describir los datos y sus relaciones más que especificar los detalles de almacenamiento.

En la fase de diseño lógico el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema que se va a utilizar posteriormente. El modelo de

implementación de los datos suele ser el modelo relacional en las bases de datos relacionales, este paso normalmente consiste en la traducción del modelo conceptual definido Entidad Relación en un esquema de relación.

Finalmente el diseñador usa el esquema de base de datos resultante en la siguiente fase de diseño físico, en la que se especifican las características físicas de la base de datos. La forma de organización de los archivos y las estructuras de almacenamiento interno.

## 3.2 Modelado de Datos

Tradicionalmente, la semántica del mundo real es capturada en un modelo de datos y es modelada a través de restricciones con el objetivo de asegurar la consistencia de los datos en la base de datos resultante.

Los modelos de datos son vehículos para describir la realidad. Los programadores usan los modelos de datos para construir esquemas, los cuales son representaciones de la realidad. La calidad de los esquemas resultantes depende no sólo de la habilidad de los programadores, sino también de las características del modelo de datos seleccionado(BATINI, 1994).

La abstracción es un proceso mental que se aplica al seleccionar algunas características y propiedades de un conjunto de objetos y excluir otras no pertinentes. En otras palabras, se hace una abstracción al fijar la atención en las propiedades consideradas esenciales de un conjunto de cosas y desechar sus diferencias.

Los modelos conceptuales deben ser buenas herramientas para representar la realidad; por esta razón, deben poseer las siguientes cualidades:

**1.- Expresividad.** Los modelos conceptuales difieren en la elección y número de las distintas estructuras del modelado que ofrecen. En general, la disponibilidad de una amplia gama de conceptos hace posible una representación más extensa de la realidad; por este motivo, los modelos más ricos en conceptos son también muy expresivos.

**2.- Simplicidad.** Un modelo conceptual debe ser simple, para que un esquema creado con ese modelo sea fácil de entender por los diseñadores y usuarios de la aplicación de la base de datos.



Aunque en muchas ocasiones la expresividad y la simplicidad son objetivos en conflicto; si un modelo es semánticamente rico es muy probable que no sea simple.

**3.- Minimalidad.** Esta propiedad se consigue cuando cada concepto presente en el modelo tiene un significado distinto con respecto a todos los demás, (en otras palabras, si ningún concepto puede expresarse mediante otros conceptos).

**4.-Formalidad.** Los esquemas creados usando modelos conceptuales de datos representan una especificación formal de los datos. La formalidad requiere que todos los conceptos del modelo tengan una representación única, precisa y bien definida.

En general, un modelo no es capaz de expresar todas las propiedades de una realidad determinada; algunas de estas propiedades deben expresarse mediante notas que complementen el esquema. Sin embargo, el número de notas necesarias puede hacerse arbitrariamente pequeño, al incorporar conceptos más expresivos en el modelo.

Modelar objetos de la realidad es un problema crucial, normalmente abierto a varias soluciones alternativas; se sabe que la misma realidad se puede modelar de formas diversas.

La razón para preocuparse por el diseño de las bases de datos se debe a que es una fase crucial para la consistencia, integridad y precisión de los datos. Si una base de datos está mal diseñada, los usuarios tendrán dificultades a la hora de acceder a ciertos tipos de información y existe el riesgo de que ciertas búsquedas puedan producir información errónea.

### 3.3 Modelos para Datos Semi-estructurados

En bases de datos semi-estructurados existen dos enfoques para su implementación, el primero captura las restricciones del mundo real en un modelo de datos y el segundo se utiliza en el caso en que un documento exista sin un esquema.

El modelo de datos que se utiliza en el diseño de esquemas para datos semi-estructurados tiene requisitos diferentes a los usados en el diseño de esquemas de bases de datos relacionales. A fin de apoyar el primer enfoque, el modelo de datos debe proporcionar un medio para modelar la instancia del documento, el documento esquema, y los atributos de identidad de conjuntos de elementos. Los conceptos fundamentales de datos semi-estructurados también deben ser parte del

modelo. Ellos incluyen la estructura jerárquica de conjuntos de elementos, y el ordenamiento de conjuntos de elementos y atributos.

Algunos modelos que se describen a continuación, son comúnmente utilizados para describir documentos XML y su estructura; con el objetivo de entender su funcionamiento los ejemplos se ilustran utilizando el documento de la Figura 3.1.

```
<inscripcion>
  <Departamento nombre = "CS">
    <curso codigo= "CS1102">
      <titulo>Estructura de Datos </titulo>
      <estudiantestuNo="stu123" >
        <estudiantestuNombre> Rodrigo </estudiante>
        <direccion>Av. 10 de Mayo </direccion>
        <calificacion> A </calificacion>
      </estudiante>
      <estudiantestuNo="stu125" >
        <estudiantestuNombre> Juan </estudiante>
        <pasatiempo> Leer </pasatiempo>
        <pasatiempo> Nadar </pasatiempo>
        <calificación> B </calificación>
      </estudiante>
    </curso>
    <curso codigo= "CS1104">
      <estudiantestuNo="stu123" >
        <estudiantestuNombre> Rodrigo </estudiante>
        <direccion>Av. 10 de Mayo </direccion>
      </estudiante>
      <estudiantestuNo="stu125" >
        <estudiantestuNombre> Juan </estudiante>
        <pasatiempo> Leer </pasatiempo>
        <pasatiempo> Nadar </pasatiempo>
      </estudiante>
    </curso>
  </Departamento>
</inscripcion>
```

**Figura 3.1** Ejemplo de Documento XML

### 3.3.1 DOM (Document Object Model)

El Modelo de Objetos del Documento (DOM) es una interfaz de programación de aplicaciones (API) para documentos válidos HTML y bien contruidos XML. Define la estructura lógica de los

documentos y el modo en que se acceden y manipulan. XML presenta a los datos como documentos, y se puede utilizar DOM para manejar estos datos(LE HÉGARET Philippe., 2004).

Como una especificación de W3C, un objetivo importante del Modelo de Objetos del Documento es proporcionar un interfaz estándar de programación, que pueda ser utilizada en una amplia variedad de entornos y aplicaciones.

En DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol; para ser más preciso, es más bien como un "bosque", que puede contener más de un árbol. Cada nodo representa un objeto que contiene a uno de los componentes de la estructura XML, los tres tipos comunes de elementos son los nodos elemento, nodos atributo y nodos texto.

Como se ilustra en la Figura 3.2, los nodos texto no tienen nombre pero contienen texto, los nodos atributo tienen ambos nombre y texto, y los nodos elemento tienen nombre y a su vez pueden tener nodos hijo. Las aristas que unen a los nodos representan las relaciones entre los nodos.

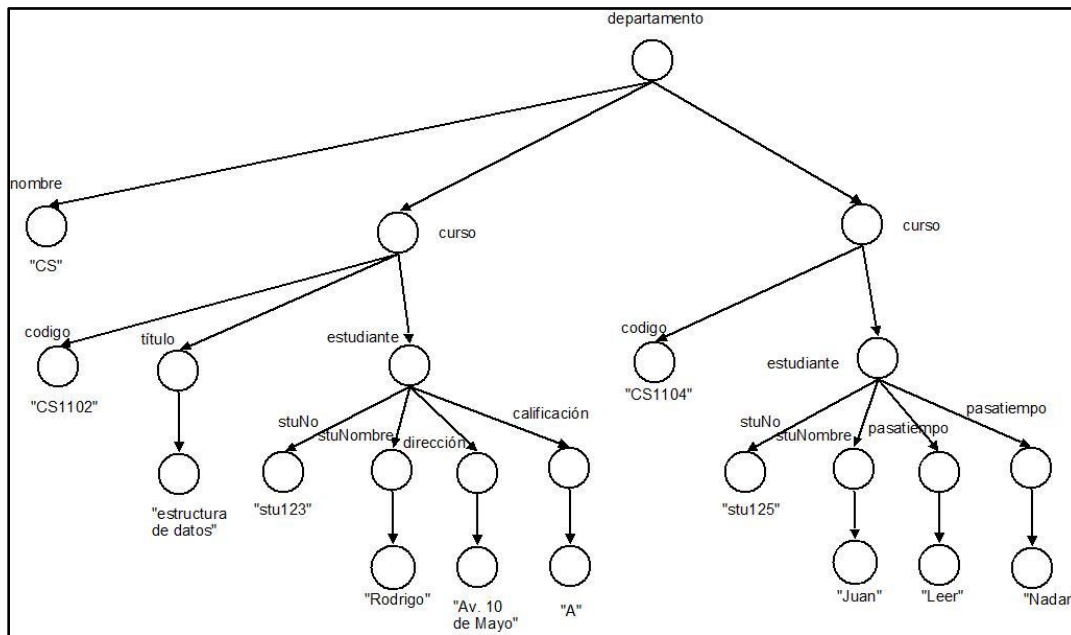


Figura 3.2 Ejemplo DOM

Un árbol DOM representa la instancia de un documento, muestra la estructura jerárquica de los elementos y las relaciones implícitas entre los elementos; es posible distinguir entre atributos y

elementos. Sin embargo DOM sólo representa la instancia de un Documento XML, no representa un esquema de información directamente, como, el grado de las relaciones y las restricciones de participación sobre el conjunto de elementos en el conjunto de relaciones, por la misma razón no es posible distinguir entre los elementos ordenados y los elementos desordenados, o si un atributo pertenece a una relación o a un elemento.

DOM no especifica que los documentos sean implementados como un árbol, ni tampoco especifica cómo deben implementarse las relaciones entre los objetos. DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente.

### 3.3.2 OEM (Object Exchange Model)

El Modelo de Intercambio de datos (OEM) representa el contenido de un documento XML. Un modelo OEM es un grafo dirigido etiquetado, donde los vértices son objetos y las aristas son relaciones (MCHUGH Jason., 1997), la Figura 3.3 ilustra el modelo de una parte del documento de la Figura 3.1.

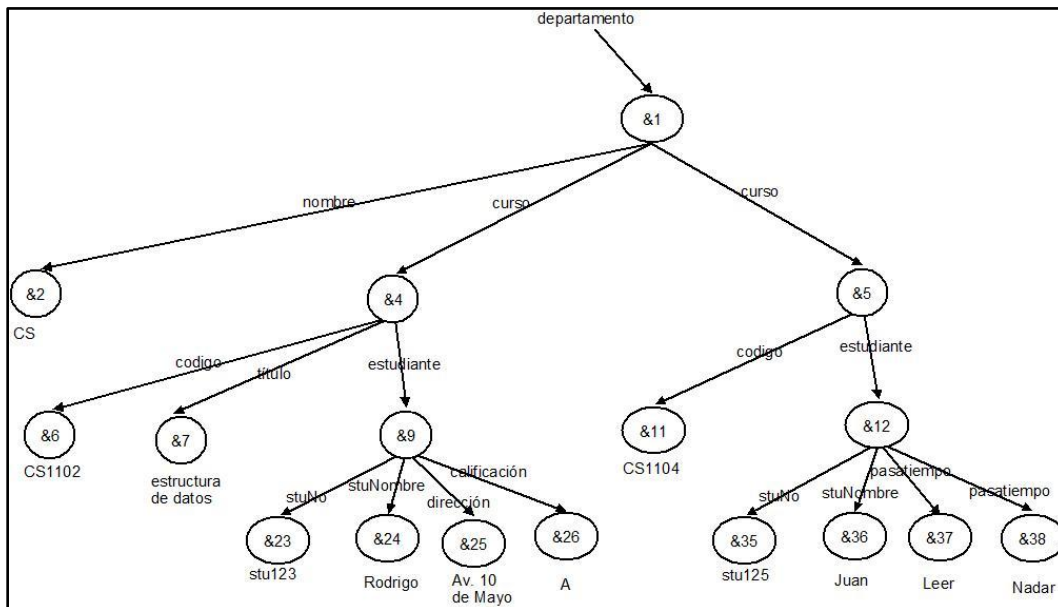


Figura 3.3 Ejemplo OEM

Cada objeto tiene un identificador objeto (OID) único, una etiqueta y un valor. Existen dos tipos de objetos: atómicos y complejos, ambos objetos son descritos en forma de tríos: (OID, etiqueta, valor). Un Objeto atómico contiene el valor de uno de los tipos básicos, por ejemplo enteros, cadenas, números reales, etc. Un objeto complejo es una composición de objetos donde su valor es un conjunto de referencias a objetos, denotado como un conjunto de pares (etiqueta, OID).

Considerando el modelo de la Figura 3.3, los nodos hoja del grafo son objetos atómicos y los nodos internos son objetos complejos, el objeto complejo con el identificador &1 y el nombre departamento es especificado en forma de trío como sigue:

(&1, departamento, {(nombre, &2), (curso, &4), (curso, &5)}),

Donde el conjunto de tuplas representan a los objetos a los que el objeto &1 hace referencia.

Los tres objetos atómicos con los objetos identificadores &27, &28 y &31 son especificados en forma de tríos como sigue:

(&23, stuNo, stu125)

(&24, stuNombre, Rodrigo)

(&26, calificación, A)

Un modelo de intercambio de datos (OEM) indica la estructura jerárquica de los objetos. Aunque proporciona una representación diagramática y textual, no sólo tiene las mismas deficiencias que DOM además sufre de no distinguir entre elementos y atributos.

### 3.3.3 S3-graph (Semi-Structured Schema Graph)

Semi-Structured Schema Graph es un grafo dirigido, donde cada nodo en el grafo puede ser clasificado como un nodo entidad o referencia. Un nodo entidad representa una entidad, la cual puede constar de tipos de datos atómicos como cadena, fecha, etc., o tipos complejos como un estudiante. Un nodo referencia es un nodo que hace referencia a otro nodo entidad (LEE, 1999).

Cada arista dirigida en el grafo es asociada a una etiqueta. Las etiquetas representan la relación entre un nodo fuente y el nodo destino, la etiqueta puede tener el sufijo “\*”; la interpretación de la etiqueta y el sufijo dependen del tipo de arista, existen tres tipos de aristas:

- **Arista Componente:** Un nodo v1 es conectado a un nodo v2 vía una arista componente, con etiqueta T si v2 es un componente de v1. Esta arista es denotada por una flecha de línea sólida. Si T tiene un sufijo “\*”, la relación es interpretada como: “el tipo de entidad representada por v1 tiene muchos T”. De otra manera, la relación es interpretada como: “el tipo de entidad representada por v1 tiene al menos un T”.
- **Aristas de Referencia:** Un nodo v1 es conectado a un nodo v2 vía una arista de referencia, si v1 hace referencia a una entidad representada por el nodo v2; Este tipo de arista se representa mediante una flecha de línea punteada.
- **Arista Raíz:** Un nodo v1 es señalado por una arista raíz con una etiqueta T, la arista es denotada por una flecha de línea solida sin ningún nodo de origen y sin sufijo para la etiqueta T.

La Figura 3.4 muestra el S3-graph para el documento de la Figura 3.1. El nodo #1 representa un nodo entidad que representa a la entidad Departamento y también es el nodo raíz.

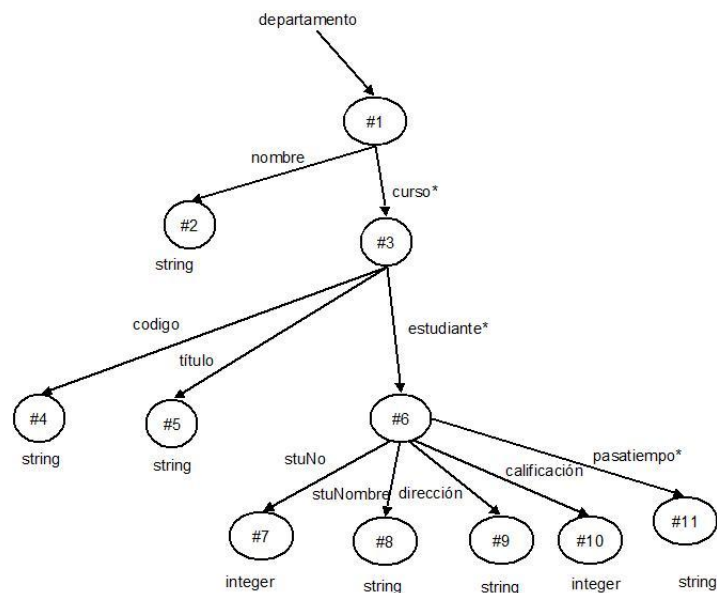


Figura 3.4 Ejemplo de S3-Graph

El nodo **#2** es un nodo entidad, representa el nombre del departamento y es un nodo hoja asociado a un tipo de dato atómico “string”, por lo que, un nombre debe ser de tipo cadena, la arista es de tipo componente entre el nodo **#1** y el nodo **#2**, es decir, cada departamento tiene al menos un nombre.

La etiqueta de la arista de tipo componente que une a los nodos **#1** y **#3** tiene el sufijo “\*”, lo que significa que un departamento tiene muchos cursos.

Este modelo captura la estructura jerárquica del conjunto de elementos y provee referencias. Sin embargo, no distingue entre los atributos de los tipos de entidad y los conjuntos de relaciones, por ejemplo, no es claro el hecho de que la calificación es un atributo de la relación entre curso y estudiante. Además, el modelo sólo es capaz de representar relaciones binarias.

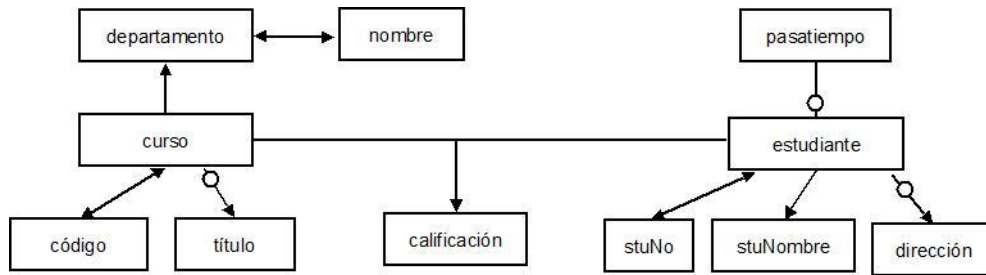
### 3.3.4 CM Hypergraph and SchemeTree

CM Hypergraph and Scheme Tree es un modelo de datos que consiste en dos diagramas, el CM Hypergraph (Modelo Conceptual) y SchemeTree (el esquema de árbol)(W. David Embley., 2001). El modelo de datos fue diseñado para representar la semántica con la elaboración de algoritmos que garantizan el desarrollo de documentos XML conservando sus propiedades.

En el modelo CM hypergraph los conjuntos de objetos son representados como rectángulos etiquetados; las relaciones son representadas por aristas y las restricciones de participación son representadas utilizando puntas de flecha y el símbolo “o” sobre las aristas. Una arista sin puntas de flecha representa una relación muchos a muchos; una arista con una punta de flecha representa una relación muchos a uno y una arista con punta de flecha en los extremos representa una relación uno a uno, el símbolo “o” indica que el objeto es opcional. Se permite representar dependencias funcionales, por ejemplo código → título, stuNo → {nombre, dirección}.

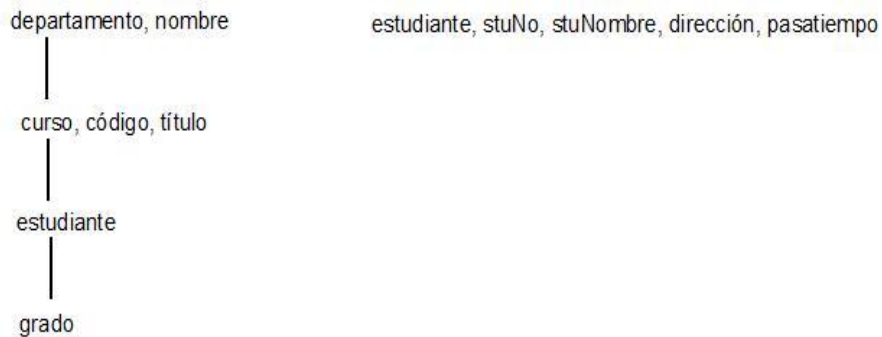
El SchemeTree representa la estructura jerárquica del conjunto de objetos, las aristas representan los elementos-subelementos de una relación. Considerando los ejemplos de la Figura 3.5 el CM Hypergraph tiene los conjuntos de objetos departamento, nombre, curso, código, título, estudiante, stuNo, stuNombre, dirección, calificación y pasatiempo. El CM hypergraph modela los

siguientes requerimientos: el departamento tiene un único nombre y uno o más cursos, el código del curso es único, un curso pertenece únicamente a un Departamento, tiene un código único y un título opcional. Existe una relación ternaria entre curso, estudiante y calificación. Cada estudiante en un curso tiene una calificación, un estudiante tiene los datos: stuNo (único), stuNombre, dirección (opcional) y tiene cero o más pasatiempos.



**Figura 3.5** Ejemplo de CM Hypergraph

El SchemeTree ilustrado en la Figura 3.6 representa la estructura jerárquica: departamento y nombre están ubicados en la raíz; curso, código y título son anidados dentro de departamento, estudiante está anidado dentro de curso, y la calificación está anidada dentro de estudiante, la información del estudiante stuNo, stuNombre, dirección y pasatiempo se encuentran separados del SchemeTree.



**Figura 3.6** Ejemplo de SchemeTree



CM hypergraphs permite modelar relaciones binarias y n-arias, pero no permite modelar el anidamiento jerárquico directamente, por lo que se auxilia del esquema de árbol, desde que este modelo no distingue entre atributos y conjuntos de objetos, el número de conjuntos de objetos llega a ser muy grande y el grafo llega a ser muy complejo. La interpretación de la restricción opcional es ambigua, por ejemplo el símbolo “o” entre la arista curso y título, si el símbolo esta cerca de título significa que el título es opcional, pero si estuviera tan cerca de curso significaría que el curso es opcional para el título, por lo que se debe tener especial cuidado en la ubicación del símbolo, otro problema con esta restricción es en las relaciones ternarias, por ejemplo, cómo representar que un estudiante está tomando un curso pero no tiene una calificación por el momento.

El SchemeTree representa las relaciones jerárquicas entre los conjuntos de objetos. Las relaciones jerárquicas binarias pueden ser modeladas directamente, y las n-arias son modeladas utilizando más de un Scheme Tree por lo que la información acerca del grado de la relación se pierde.

### 3.3.5 EER (Entity Relationship diagram) y XGrammar

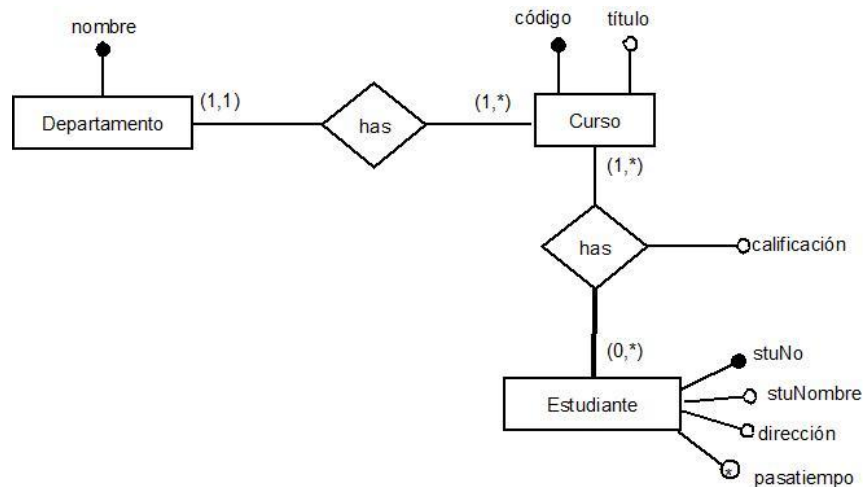
EER (Entity Relationship diagram) y XGrammar son un diagrama y un lenguaje respectivamente, utilizado para modelar esquemas XML(MURALI Mani., 2001). El lenguaje XGrammar fue diseñado con el propósito de capturar las más importantes características del esquema XML. El diagrama EER difiere de las otras notaciones de diagramas, desde el momento en que captura todos los conceptos que pueden ser capturados en el modelo Entidad Relación, y captura también las relaciones jerárquicas y el orden de los conjuntos de elementos.

Las relaciones jerárquicas son representadas a través de una relación intermedia con la etiqueta “has”. El orden en los elementos se expresa con una línea sólida entre el conjunto de la relación y el conjunto de la entidad ordenada. Los conjuntos de entidades son representados como rectángulos y el conjunto de relaciones se representa a través de diamantes en las aristas.

Considerando el ejemplo de la Figura 3.7 con los conjuntos de entidades Departamento, Curso y Estudiante; la entidad Departamento tiene un atributo llave “nombre”; el atributo llave de Curso

es “código” y tiene un atributo simple “título”; el Estudiante tiene un atributo llave “stuNo”, y atributos simples “stuNombre” y “dirección”, así como un atributo multivaluado “pasatiempo”; existen dos conjuntos de relaciones con la etiqueta “has” que representan la relación jerárquica entre los conjuntos de entidades Departamento y Curso, otra entre Curso y Estudiante, esta última tiene un atributo “calificación”. Un departamento tiene uno o más cursos y un curso pertenece a un único departamento; un curso tiene cero o más estudiantes y un estudiante pertenece a uno o más cursos.

El orden de las entidades es representado por una línea gruesa en el diagrama EER. El hecho de que un estudiante deba tomar un curso, debe ocurrir en un orden particular, el cual se representa en la Figura 3.7.



**Figura 3.7** Ejemplo de EER (Entity Relationship diagram)

XGrammar permite expresar la relación jerárquica entre los conjuntos de entidades, distingue entre atributos y elementos, representa restricciones de participación en los elementos hijo, y representa referencias. Una definición de XGrammar referente al esquema de la Figura 3.7 se describe en la Figura 3.8. El lenguaje XGrammar define los conjuntos de entidades y las restricciones establecidas en ellas en forma de quintuplas {N, T, S, E, A} donde:

**N** es un conjunto de símbolos no terminales, los cuales representan conjuntos de entidades.

**T** es un conjunto de símbolos terminales, los cuales representan instancias de los conjuntos de entidades y atributos.

**S** es un símbolo no terminal que representa la raíz del documento.

**E** es un conjunto de reglas de producción, las cuales describen la relación entre los conjuntos de entidades.

**A** es un conjunto de reglas de producción las cuales describen atributos.

Las reglas de producción en **E** y **A** expresan las restricciones de interés. El autor usa la notación  $\epsilon$ ,  $\rightarrow$  y  $@$  para expresar un subelemento vacío, una referencia y un atributo respectivamente.

```
N = (Departamento, Curso, Estudiante, Has)
T = (departamento, curso, estudiante, has, nombre, código, título, calificación,
stuNo, stuNombre, dirección, pasatiempo, estudianteRef)
S = (Departamento)
E = (Departamento  $\rightarrow$  departamento(Curso+),
Curso  $\rightarrow$  curso(Has*),
Has  $\rightarrow$  has ( $\epsilon$ ),
Estudiante  $\rightarrow$  estudiante( $\epsilon$ ))

A = (Departamento  $\rightarrow$  departamento(@nombre::string),
Curso  $\rightarrow$  curso(@código::string, @título?::string),
Has  $\rightarrow$  has (@estudianteRef::IDREF  $\rightarrow$  Estudiante, @calificación?::string),
Estudiante  $\rightarrow$  estudiante(@stuNo::string, @stuNombre::string, @dirección?::string,
@pasatiempo*::string)
```

**Figura 3.8** Ejemplo de XGrammar

Considerando la definición XGrammar de la Figura 3.8, el conjunto **N** contiene los nombres de los conjuntos de entidades: Departamento, Curso y Estudiante, así como el conjunto de entidades has. El conjunto de relaciones “has” entre los conjuntos de entidades Curso y Estudiante es modelado como un conjunto de entidades, debido a que tiene el atributo calificación; justo como en el modelo relacional donde las relaciones muchos a muchos que tienen atributos, son capturadas en una relación aparte, XGrammar modela este tipo de relaciones de manera separada, la relación entre Departamento y Curso es uno a muchos y es capturada anidando Curso dentro de Departamento. El conjunto **T** contiene las entidades y atributos, **S** contiene el elemento raíz del documento, el conjunto **E** especifica los conjuntos de relaciones en los conjuntos de entidades, la primera regla en **E** especifica que el conjunto de entidades Departamento tiene uno o más Cursos, Departamento es un conjunto de entidades, mientras departamento es una instancia de Departamento, la segunda regla especifica que una entidad perteneciente al conjunto de entidades Curso tiene cero o más entidades del conjunto entidades has como subelemento, la

tercera y cuarta regla especifican que el conjunto entidades Has y Estudiante no tienen elementos hijos.

En el conjunto **A** el símbolo @ denota un atributo. El conjunto de entidades Departamento tiene un atributo nombre el cual es de tipo string, el conjunto entidades Curso tiene dos atributos código y título, los cuales son opcionales, el conjunto entidades Has tiene el atributo studenRef, el cual es una referencia al conjunto de entidades Estudiante, calificación es un atributo opcional, el conjunto entidades Estudiante tiene los atributos stuNo, stuNombre, dirección que es opcional y el atributo pasatiempo el cual es multivalorado.

EER y XGrammar son modelos que sirven para diferentes propósitos y pueden representar distintos conceptos. Sin embargo, con el diagrama EER no es posible representar qué conjunto de entidades es la raíz del documento. Existe un problema con la representación de la estructura jerárquica del esquema Semi-Estructurado en el diagrama EER; el conjunto de relaciones “has” es utilizado para expresar la estructura jerárquica, pero esta relación al no tener un direccionamiento, no es claro distinguir, qué conjunto de entidades es el elemento y cuál es el subelemento, por lo que la relación no puede representar la estructura jerárquica directamente. La estructura jerárquica puede ser representada en XGrammar sólo como relaciones binarias.

Es posible representar las restricciones de participación de padres e hijos en el diagrama EER, pero no en XGrammar; es posible identificar a un atributo como una llave en el diagrama EER, pero no es posible mostrar si otros atributos son obligatorios u opcionales. Una forma de superar este problema es representar atributos como los conjuntos de entidades, pero esto podría llevar a que el conjunto de entidades crezca. No es posible representar e identificar atributos en XGrammar.

No hay distinción entre atributos de un conjunto de entidades y atributos de un conjunto de relaciones en XGrammar; tampoco es posible representar el orden en los atributos. El concepto de atributo es el mismo que en el modelo Entidad Relación lo cual difiere el significado del concepto de atributo en documentos XML, por lo que algunos atributos en EER pueden ser modelados como elementos en los documentos XML.

### 3.4 Diseño de Bases de Datos XML

Un sistema de base de base de datos bien definido se basa en un modelo de datos bien definido. La complejidad de los repositorios de datos relacionados con XML y la necesidad de integrar el manejo de documentos estructurados con el manejo de otros tipos de datos, crea un reto especial para el modelo de datos subyacente de los documentos XML, siendo este un DTD o un XML Schema. Algunas veces, los elementos son ordenados y otras veces no. Este tipo de modelo simplificado puede ser suficiente para el desarrollo de las capacidades relativas a la estructura jerárquica de los elementos. Sin embargo para ser capaz de manejar documentos XML como una base de datos, se requiere un modelo de datos más rico(SALMINEN, 2001).

A diferencia de las bases de datos convencionales, los datos en un documento de base de datos no representan al negocio directamente. En su lugar representan una colección de documentos, que capturan la información del negocio. El modelo de datos debe soportar la descripción de documentos a medida que son construidos. Para una base de datos XML la cuestión semántica es fundamental.

El diseño de bases de datos XML no tiene una metodología tan aceptada como en el relacional. Una base de datos XML es generalmente modelada a través de árboles o grafos dirigidos, con nodos etiquetados, representando a cada documento como un árbol. El diseño de bases de datos XML sería sencillo si se pudiera aplicar el modelo Entidad Relación o el Relacional directamente, sin embargo XML es semi-estructurado y con características muy flexibles lo que hace que estos modelos sean inservibles, debido a que, en XML los datos son inherentemente ordenados permiten diferentes estructuras, tipos complejos como opcionales y elementos multivalorados, no hay una manera simple de especificar relaciones muchos a muchos por su estructura jerárquica.

Después de identificar los requerimientos es necesario identificar la semántica de los datos y las relaciones entre ellos, una vez que la semántica ha sido identificada, el siguiente paso sería especificar la estructura de los datos, para las bases de datos XML esto significa especificar elementos, atributos, tipos, valores, espacios de nombres y un contenedor de relaciones.

Los diseñadores necesitan considerar una estructura formal y expresividad en el esquema del lenguaje como un DTD y un esquema XML. Cuando se utiliza un motor más potente en el esquema de lenguaje, como un XML Schema, los diseñadores necesitan elegir un patrón de modelo de diseño de esquema.

## Capítulo 4. Modelo de Base de Datos Semi-estructurados propuesto

---

Un modelo de datos es una colección de herramientas conceptuales para la descripción de datos, las relaciones entre ellos, su semántica y las restricciones de consistencia; el aspecto semántico yace en la representación del significado de los datos. Los modelos de datos ofrecen un modo de describir el diseño de las bases de datos.

Un modelo de datos Semi-Estructurados permite la especificación de datos, donde los elementos de datos individuales del mismo tipo pueden tener diferentes tipos de atributos.

El modelo propuesto, es un modelo de alto nivel que toma algunas características del modelo Entidad Relación, esto con el objetivo de que los diseñadores de bases de datos Relacionales puedan entender el concepto del mismo, fácilmente; se basa en la percepción del mundo real, que consiste en una colección de entidades, atributos y relaciones entre entidades. La principal diferencia con un modelo para diseño de bases de datos relacional es el anidamiento de entidades, las restricciones de participación en atributos y entidades.

El modelo proporciona una herramienta para facilitar el diseño de bases de datos Semi-estructurados, ya que si, el modelo formulado representa correctamente las restricciones del problema que se está modelando, aplicando un algoritmo de transformación a dicho modelo, se obtiene la DTD que permita la validación del contenido de los documentos XML almacenados en la bases de datos.

El modelo consta de los siguientes componentes:

- Entidades.
- Atributos.
- Restricciones de Participación de Atributos.
- Relaciones.
- Restricciones de Participación de Entidades en una Relación.
- Referencias.

## Conjuntos de Entidades

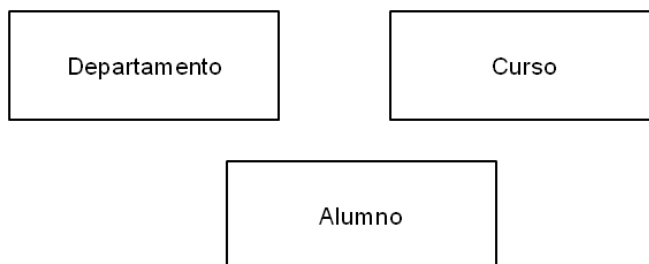
Las entidades modelan «cosas» del mundo real, que son distinguibles de todas las demás entidades. Una entidad tiene un nombre y un conjunto de atributos que pertenecen a dicha entidad, aunque, debido a la naturaleza de los datos Semi-Estructurados, las entidades son caracterizadas por su nombre, y no por sus atributos. Una entidad puede ser concreta, como una persona o un libro, o puede ser abstracta, como un préstamo, unas vacaciones o un concepto. Las entidades son representadas mediante un rectángulo etiquetado con el nombre de la entidad.

Un conjunto de entidades agrupa entidades del mismo tipo que comparten propiedades o atributos. El conjunto de todos los alumnos inscritos en una institución, por ejemplo, se pueden definir como el conjunto de entidades alumno. Las entidades individuales que constituyen un conjunto se llaman instancias del conjunto de entidades. Así, cada uno de los alumnos de una institución son instancias del conjunto de entidades alumno.

```
< Departamento deptoNombre=Ciencias de la Computación>
  <Curso deptoPrefijo= "CS" cursoNo="101">
    <tituloCurso> Principios de Programación </tituloCurso>
    <Alumno NoCuenta="123456">
      <nombreAlumno>
        <nombre> Yolanda </nombre>
        <apellidoPaterno> Zamora <apellidoPaterno>
        <apellidoPaterno> Espinoza <apellidoPaterno>
      </nombreAlumno>
      <Calificacion> 10 </Calificacion>
      <Telefono> Casa 5534415678 </Telefono>
      <Telefono> Oficina 7654244 </Telefono>
    </Alumno>
  </Curso>
  <Curso deptoPrefijo= "CS" cursoNo="210">
    <tituloCurso> Sistemas Computacionales I </tituloCurso>
    <Alumno NoCuenta="234567">
      <nombreAlumno>
        <nombre> Juan </nombre>
        <apellidoPaterno> Robles <apellidoPaterno>
        <apellidoPaterno> Morales <apellidoPaterno>
      </nombreAlumno>
    </Alumno>
    ....
  </Curso>
</Departamento>
```

**Figura 4.1** Documento XML

Considerando el ejemplo ilustrado en la Figura 4.1 que corresponde a un documento XML sencillo, el cual, muestra información sobre los alumnos del departamento de Ciencias en la Computación; los conjuntos de entidades en el documento son Departamento, Curso y Alumno, las cuales son representadas gráficamente en la Figura 4.2; de acuerdo al documento de la Figura 4.1 las instancias de la entidad Alumno, son los alumnos: Yolanda Zamora Espinoza y Juan Robles Morales.



**Figura 4.2** Representación gráfica de las entidades Departamento, Curso y Alumno

## Atributos

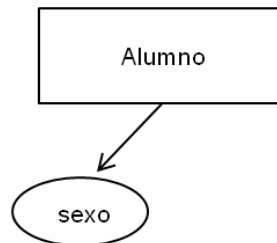
Los atributos son propiedades descriptivas que posee cada miembro de un conjunto de entidades o relaciones. En un modelo de datos Semi-estructurados, el conjunto de atributos asociado con una entidad proporciona información adicional de una entidad, por ejemplo, el nombre de un alumno es un atributo; no se espera que todas las instancias de una entidad tengan el mismo número de atributos, debido a esto, los atributos de las instancias de las entidades son heterogéneos. Cada instancia de una entidad tiene un valor para cada atributo y para cada atributo existe un conjunto de valores permitidos. Los atributos pueden ser de los siguientes tipos:

- **Atributos Monovalorados:** Son atributos que pueden tener sólo un valor, estos atributos deben tener un valor para todas las instancias de una entidad; por ejemplo, un alumno sólo puede tener un número de cuenta, con el cual se va a identificar en la institución, y todos los alumnos dentro de la institución deben contar con esta información. En ocasiones es necesario colocar en estos atributos un valor fijo, que será el mismo para



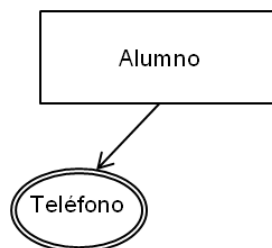
todas las instancias de una entidad y no podrá ser modificado; o un valor por default en caso de que no se especifique un valor para el atributo.

Los atributos monovalorados son representados mediante una elipse etiquetada con el nombre del atributo, y unida mediante una flecha a la entidad a la que corresponde el atributo; un ejemplo de un atributo monovalorado, es el sexo del alumno, el cual se ilustra en la Figura 4.3.



**Figura 4.3** Representación gráfica del atributo monovalorado sexo.

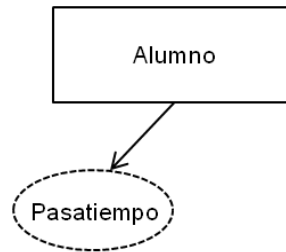
- **Atributos Multivalorados:** Son atributos que pueden tener más de un valor. Este tipo de atributos se representa mediante una elipse doble etiquetada con el nombre del atributo, y unida mediante una flecha a la entidad a la que corresponde. Por ejemplo, un alumno puede tener uno o más números de teléfono como se ilustra en la Figura 4.4.



**Figura 4.4** Representación gráfica del atributo multivalorado Teléfono.

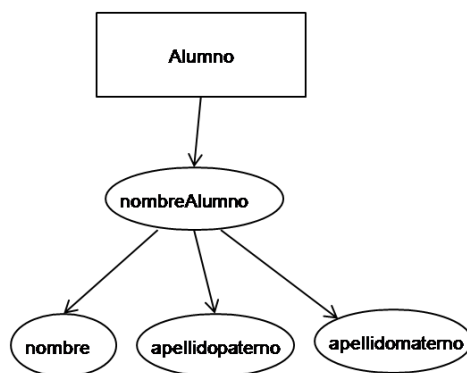
- **Atributos Opcionales:** Son atributos que pueden no aparecer en algunas instancias de la entidad; este tipo de atributo a la vez puede ser de multivalorado cuando se requiera que el atributo tome más de un valor. Por la naturaleza semi-estructurada de los datos, no se espera que todas las instancias de una entidad tengan los mismos atributos. Los atributos

opcionales son representados mediante una elipse de línea punteada, etiquetada con el nombre del atributo y unida mediante una flecha a la entidad a la que corresponde; un ejemplo de éstos se ilustra en la Figura 4.5, donde el atributo pasatiempo de la entidad alumno puede aparecer en algunas instancias y puede no aparecer en otras, esto en el caso donde un alumno no tenga un pasatiempo, así mismo, en algunas instancias el alumno puede tener más de un pasatiempo.



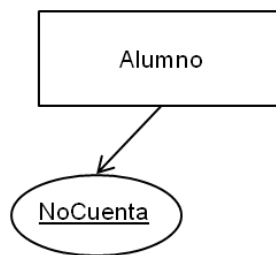
**Figura 4.5** Representación gráfica del atributo opcional pasatiempo.

- **Atributos Compuestos:** Son atributos que se componen de otros atributos; para representar un atributo compuesto, se coloca una elipse etiquetada con el nombre del atributo principal, y unidas a esta mediante flechas, las elipses correspondientes a los atributos que la componen; un ejemplo de este tipo de atributo se ilustra en la Figura 4.6, donde el atributo compuesto, nombreAlumno de la entidad alumno, se compone de los atributos: nombre, apellido paterno y apellido materno.



**Figura 4.6** Representación gráfica del atributo compuesto nombre.

- **Atributos Llave.** Son atributos que toman valores únicos para cada instancia de un conjunto de entidades dentro de un documento, en otras palabras, no se permite que dos o más instancias de una entidad en un documento, tengan exactamente los mismos valores en sus atributos. El atributo llave es elegido por el diseñador de la base de datos, como el elemento principal que identifica a las instancias y se representa subrayando el nombre del atributo. El atributo llave se debería elegir de manera que nunca o muy raramente cambie. Un ejemplo de este tipo de atributos, es el Número de Cuenta de un alumno, el cual se representa en la Figura 4.7.



**Figura 4.7** Representación gráfica del atributo llave NoCuenta.

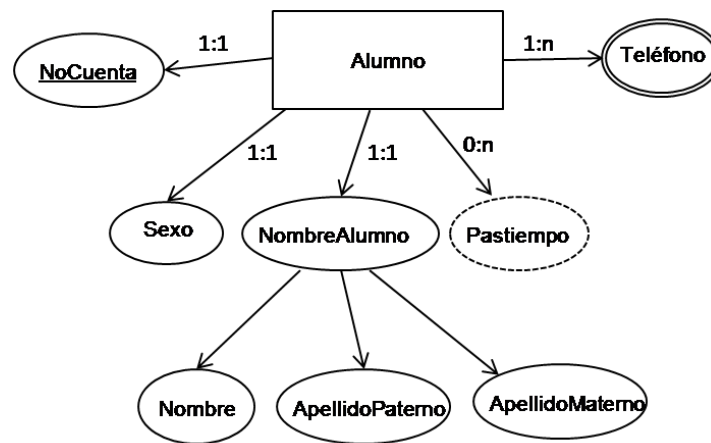
### Restricciones de participación en los atributos

Las restricciones de participación de los atributos en las entidades, se refieren, al número de valores permitidos que puede tener un atributo, en una instancia de una entidad, en el modelo se establecen las restricciones de participación para los atributos, con la finalidad de poder expresar restricciones del mundo real.

- **Atributos Monovalorados:** La restricción de participación para estos atributos es: 1:1, lo que significa, que deben tener sólo un valor, para cada instancia de la entidad; no es necesario indicar la restricción de participación del atributo en el modelo, ya que su representación gráfica indica que es un atributo monovalorado; en caso de indicar la restricción de participación, se hace mediante una etiqueta con el valor 1:1 en el arista que une al atributo con la entidad a la que corresponde.

- **Atributos Multivalorados:** La restricción de participación para estos atributos es: 1:n, es decir, las instancias de una entidad determinada, deben tener desde uno a hasta un número indefinido de valores para este atributo, y sean instancias válidas; no es necesario indicar la restricción de participación del atributo, debido a que la representación gráfica del mismo expresa la restricción de participación al ser multivalorado y no opcional, sin embargo, en caso de que se desee indicar, se coloca una etiqueta con el valor 1:n en el arista que une al atributo con la entidad a la que corresponde.
- **Atributos Opcionales:** La restricción de participación para este tipo de atributo puede ser: 0:1 ó 0:n, es decir, si se especifica: 0,1, indica que el atributo puede no aparecer en algunas instancias y en caso de aparecer puede tomar sólo un valor; si se especifica 0,n, significa, que puede no aparecer en algunas instancias y en caso de aparecer puede tener uno o más valores. La restricción de participación de este tipo de atributos, se indica colocando una etiqueta con el valor: 0:1 ó 0:n, según sea el caso, en la arista que une al atributo con la entidad a la que corresponde.
- **Atributo Llave:** La restricción de participación para estos atributos es: 1:1, la diferencia con los atributos monovalorados, es el hecho de que los atributos llave deben tener un valor diferente en cada una de las instancias de la entidad; no es necesario indicar la restricción de participación del atributo, debido a que la representación gráfica del mismo, expresa que el atributo debe tener sólo un valor al ser llave, sin embargo, en caso de que se desee indicar, se coloca una etiqueta con el valor 1:1 en el arista que une al atributo con la entidad a la que corresponde.
- **Atributos Compuestos:** Un atributo compuesto puede ser a su vez un atributo monovalorado, multivalorado u opcional, es decir puede tener restricción de participación 1:1, 1:n, 0:1, 0:n; se debe indicar la restricción de participación a través de una etiqueta en el arista que conecta el atributo con la entidad a la que corresponde.

La Figura 4.8 ilustra las restricciones de participación de los atributos de la entidad Alumno, en la representación gráfica se puede observar, que cada instancia de la entidad debe tener un valor único para el atributo llave NoCuenta, indicado con la etiqueta 1:1; el atributo NombreAlumno es un atributo compuesto, el cual se compone de los atributos: nombre, apellidoPaterno y apellidoMaterno, la etiqueta 1:1 en la arista, que une a la entidad con el atributo principal, indica que cada instancia de la entidad alumno debe tener sólo un nombre; la etiqueta en la arista que une al atributo Sexo con la entidad, indica una restricción de participación 1:1, lo que significa que en todas las instancias de la entidad debe existir un valor para el atributo; la entidad alumno tiene un atributo Teléfono con restricción de participación 1,n, indicando que una instancia de la entidad puede tener desde uno hasta indeterminados números telefónicos; por último el atributo Pasatiempo tiene una restricción de participación 0:n, indicando que en algunas instancias puede no aparecer este atributo o aparecer y tener un número indeterminado de valores, esto dependiendo del número de pasatiempos que tenga un alumno.



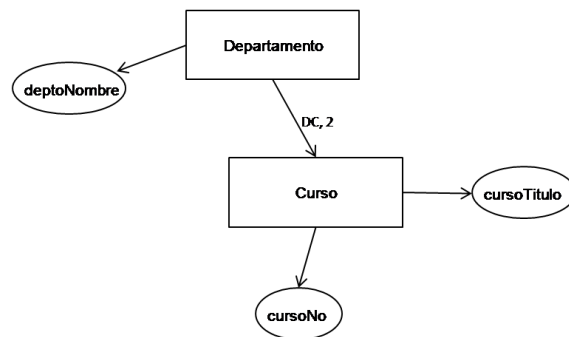
**Figura 4.8** Representación de restricciones de participación de los atributos de la entidad Alumno.

## Relaciones

Una relación es una asociación entre dos entidades; en el caso de bases de datos semi-estructurados, representan el anidamiento en los documentos. Una relación puede tener también

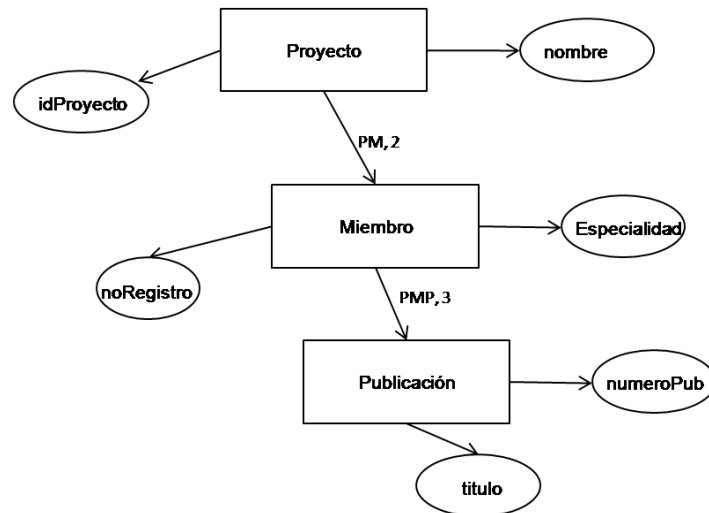
atributos, los cuales son denominados atributos descriptivos. El grado de la relación es el número de entidades que participan en la relación, las cuales pueden ser: binarias, ternarias o n-arias.

**Relación Binaria:** Relaciona dos entidades, se representa a través de dos entidades unidas por una arista dirigida que va desde la entidad padre hacia la entidad hijo; sobre la arista se coloca una etiqueta con el nombre de la relación y el grado de la misma. En la figura 4.9 se ilustra una relación binaria entre la entidad padre Departamento y la entidad hijo Curso, indicando en la etiqueta el nombre: DC y el grado: 2.



**Figura 4.9** Representación gráfica de la relación binaria entre las entidades Departamento y Curso.

**Relación Ternaria:** Es una relación entre tres entidades. Con la finalidad de mantener el anidamiento que caracteriza a los datos semi-estructurados, en una relación ternaria existe una relación binaria entre dos entidades, y otra relación entre la relación binaria y otra entidad. En la Figura 4.10 se ilustra una relación ternaria entre las entidades: Proyecto, Miembro y Publicación; existe una relación binaria entre la entidad padre Proyecto y la entidad hijo Miembro, unidas mediante una arista etiquetada con el nombre: PM y grado: 2, posteriormente, la entidad Miembro es unida a la entidad hijo Publicación mediante una arista etiquetada con nombre: PMP y grado: 3; la finalidad de esta relación ternaria es modelar las publicaciones que son escritas por un miembro que trabaja en un proyecto en específico.



**Figura 4.10** Representación gráfica de la relación ternaria entre las entidades Proyecto, Miembro y Publicación.

**Relaciones n-arias:** Una relación n-aria es una relación anidada entre ‘n’ entidades, donde la entidad hijo está relacionada a n-1 entidades, la relación se construye a través de relaciones binarias dependiendo del grado.

### Restricciones de Participación de Entidades en una Relación

En el modelo se definen las restricciones a las que el contenido de la base de datos se debe adaptar, la cardinalidad expresa el número de entidades hijo a las que puede estar asociada una entidad padre, y el número de entidades padre a las que se asocia una entidad hijo vía una relación. La cardinalidad en el modelo está dada por restricciones de participación, las cuales se aplican a las entidades padres y a las entidades hijo en una relación.

La correspondencia de cardinalidades apropiada para un conjunto de relaciones particular, depende de la situación del mundo real que el conjunto de relaciones modela.

La notación para expresar las restricciones de participación en una relación es la siguiente:

**Restricción de participación de la entidad padre**, en el contexto de una relación, expresa el número de instancias de la entidad padre a las que puede estar asociada una instancia de la entidad hijo; la restricción de participación se indica colocando una etiqueta en el extremo de la arista que une a las entidades, del lado de la entidad padre, el valor que puede tomar la restricción de participación es uno de los siguientes:

**(1:1)**, indica que una instancia de la entidad hijo sólo puede estar asociada a sólo a una instancia de la entidad padre.

**(1:n)**, indica que una instancia de la entidad hijo tiene que estar asociada mínimo a una, y máximo a “n” instancias de la entidad padre. Considerando que una relación se expresa como un anidamiento, no puede existir una instancia de una entidad hijo que no esté asociada a una instancia de la entidad padre. En caso de modelar una relación con esta restricción de participación, al final el resultado será un documento XML donde los datos de la instancia de la entidad hijo serán repetidos en cada una de las instancias de la entidad padre a las que está asociada, provocando redundancia de información.

**Restricción de participación de la entidad hijo**, indica el número de instancias de la entidad hijo que pueden estar asociadas a una instancia de la entidad padre en una relación, la restricción de participación se indica colocando una etiqueta en el extremo de la arista que une a las entidades, del lado de la entidad hijo, el valor que puede tomar la restricción de participación es una de las siguientes:

**(1:1)**, indica que una instancia de la entidad padre debe tener asociada una instancia de la entidad hijo.

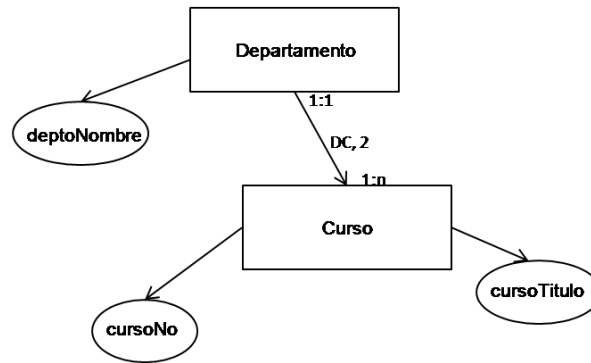
**(0:n)**, indica que una instancia de la entidad padre puede tener asociadas 0 ó un número indeterminado de instancias de la entidad hijo.

**(1:n)**, indica que una instancia de la entidad padre debe tener asociada desde una hasta un número indefinido de instancias de la entidad hijo.

La Figura 4.11 ilustra un ejemplo de las restricciones de participación de la relación binaria entre la entidad padre Departamento y la entidad hijo Curso, la restricción de participación 1:1, de la



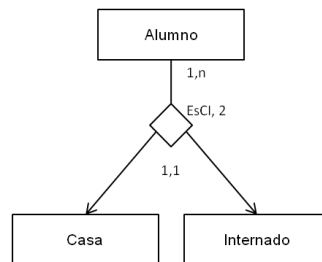
entidad Departamento expresa que cada Curso debe estar asociado a un Departamento en particular y la restricción de participación de la entidad Curso 1:n, indica que cada Departamento puede tener asociados de uno a un número indefinido de Cursos.



**Figura 4.11** Representación gráfica de las restricciones de participación de la relación binaria entre la entidad padre Departamento y la entidad hijo Curso.

### Disyunción de Entidades

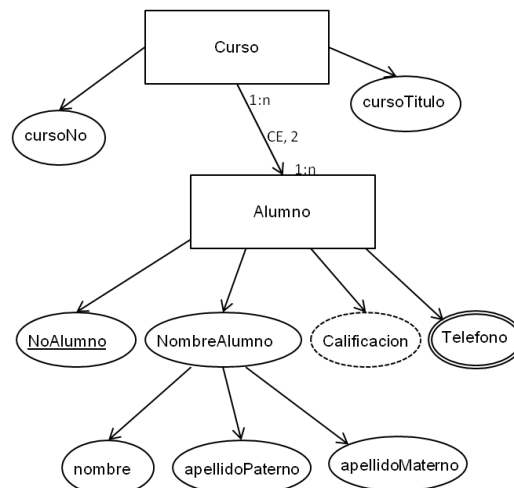
Una característica de los datos semi-estructurados es que las entidades y atributos son menos homogéneos que en un modelo de datos estructurados, permitiendo expresar disyunciones. Una disyunción es un tipo de relación, lo que permite que se puedan aplicar las restricciones de participación antes mencionadas. La disyunción de entidades permite hacer una elección entre dos o más opciones; se representa mediante un rombo, el cual une a la entidad padre con las entidades a elegibles a través de una arista. La Figura 4.12 representa una disyunción de entidades, que trata del lugar donde puede vivir un alumno, siendo posible que viva el internado de la escuela o en casa propia. La restricción de participación se lee de la siguiente manera: un alumno puede vivir en una casa o en el internado; y en una casa o en el internado, pueden vivir varios alumnos.



**Figura 4.12** Representación gráfica de la disyunción de entidades

## Referencias

Una referencia es utilizada para reducir redundancia de datos en los documentos XML resultantes del modelo. En los casos donde en el modelo existen relaciones con restricción de participación del padre 1,n, la información de la instancia de la entidad hijo se duplica en las instancias de la entidad padre a las que esté asociada, provocando redundancia de información en los documentos, esto da pie a la posibilidad de introducir inconsistencia de información; por ejemplo, si existe redundancia de información se debe cuidar que al realizar una operación de actualización, adición o borrado de información, en una instancia, los cambios se vean reflejados en todos los lugares donde la información esté replicada, de otra manera, la información replicada resultaría inconsistente. Un ejemplo de un documento XML donde se duplica la información de un Alumno en un Curso, se ilustra en la Figura 4.14, el documento es el resultado del diagrama de la Figura 4.13, donde se modela una relación binaria con restricción de participación 1,n en las entidades Curso y Alumno, lo que significa que, una instancia de la entidad Alumno puede estar asociada a una o más instancias de la entidad Curso; los datos de un Alumno se replican en cada curso al que esté inscrito. Este tipo de modelado puede causar inconsistencias, por ejemplo, si se realiza una operación alterando la información del alumno Juan en el curso de Introducción a la Computación y no se realiza la misma acción a la información correspondiente al curso Lenguajes de Programación, la información del Alumno quedaría en un estado inconsistente.



**4.13** Modelado de las entidades Curso y Alumno.

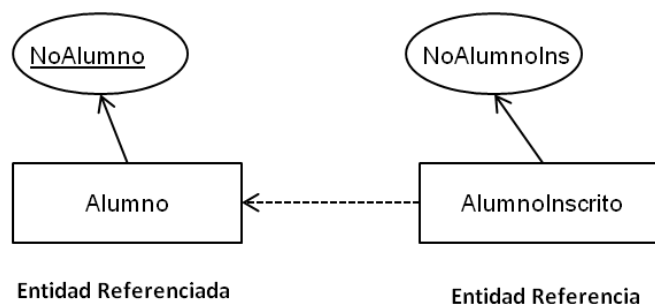
```

<Departamento deptoNombre="Ciencias de la Computación">
  <Curso cursoNo="101">
    <cursoTitulo> Introducción a la computación</cursoTitulo>
    <AlumnoNoAlumno="A10">
      <NombreAlumno>
        <nombre>Juan</nombre>
        <apellidoPaterno>Morales</apellidoPaterno>
        <apellidoMaterno>Salas</ApellidoMaterno>
      </NombreAlumno>
      <Telefono>casa 55357689</Telefono>
      <Telefono>celular 55228699</Telefono>
    </Alumno>
    <AlumnoNoAlumno="B200">
      <NombreAlumno>
        <nombre>Guadalupe</nombre>
        <apellidoPaterno>Mendoza</apellidoPaterno>
        <apellidoMaterno>Brindiz</ApellidoMaterno>
      </NombreAlumno>
      <Telefono>casa 44789065</Telefono>
      <Calificacion>9</Calificacion>
    </Alumno>
    .
    .
  </Curso>
  <Curso cursoNo="105">
    <cursoTitulo> Lenguajes de programación</cursoTitulo>
    <AlumnoNoAlumno="A10">
      <NombreAlumno>
        <nombre>Juan</nombre>
        <apellidoPaterno>Morales</apellidoPaterno>
        <apellidoMaterno>Salas</ApellidoMaterno>
      </NombreAlumno>
      <Telefono>casa 55357689</Telefono>
      <Telefono>celular 55228699</Telefono>
      <Calificacion>9</Calificacion>
    </Alumno>
    <AlumnoNoAlumno="A134">
      <NombreAlumno>
        <nombre>Felipe</nombre>
        <apellidoPaterno>Cornejo</apellidoPaterno>
        <apellidoMaterno>Salcedo</ApellidoMaterno>
      </NombreAlumno>
      <Telefono>celular 55789066</Telefono>
    </Alumno>
    .
    .
  </Curso>
</Departamento>

```

**Figura 4.14** Documento XML con información redundante

Una referencia se representa a través de una flecha de línea punteada dirigida, que va desde la entidad que hace la referencia a la entidad referenciada. Para hacer una referencia se debe tener: un atributo llave en la entidad referenciada y un atributo monovalorado en la entidad que hace la referencia. La referencia se da, cuando el valor de una instancia de la entidad que hace la referencia coincide con el valor de alguna de las instancias de la entidad referenciada. En una referencia, la instancia de la entidad que hace la referencia, heredará los atributos de la instancia de la entidad referenciada. La Figura 4.15 ilustra una referencia, donde la entidad AlumnoInscrito (entidad donde se encuentran los alumnos inscritos a un curso), hace referencia a la entidad Alumno (entidad donde se encuentran todos los Alumnos de la institución), por medio de los atributos NoAlumnoIns y NoAlumno respectivamente.



**Figura 4.15** Representación gráfica de una Referencia.

El diagrama ilustrado en la Figura 4.13, introduce redundancia de información en los documentos, con el objetivo de reducir ésta, se opta por modelar la relación como una referencia. La figura 4.16 muestra el diagrama resultante de modelado, utilizando referencias, una instancia de la entidad AlumnoInscrito (entidad que contiene a los alumnos inscritos), hace referencia a una de las instancias de la entidad Alumno (entidad que contiene a todos los alumnos de la institución), a través del atributo monovalorado: NoAlumnoIns, y el atributo llave: NoAlumno. Modelar el problema de esta manera, dará como resultado un documento XML, como el que se ilustra en la Figura 4.17, donde las instancias de la entidad Curso no incluyen la información de los alumnos, sólo se coloca el valor del atributo identificador que deben coincidir con el valor llave de una de las instancias de la entidad a la que se hace referencia, de esta manera no se replica la información, como es el caso del Alumno con identificador A10 de nombre Juan Morales Salas.

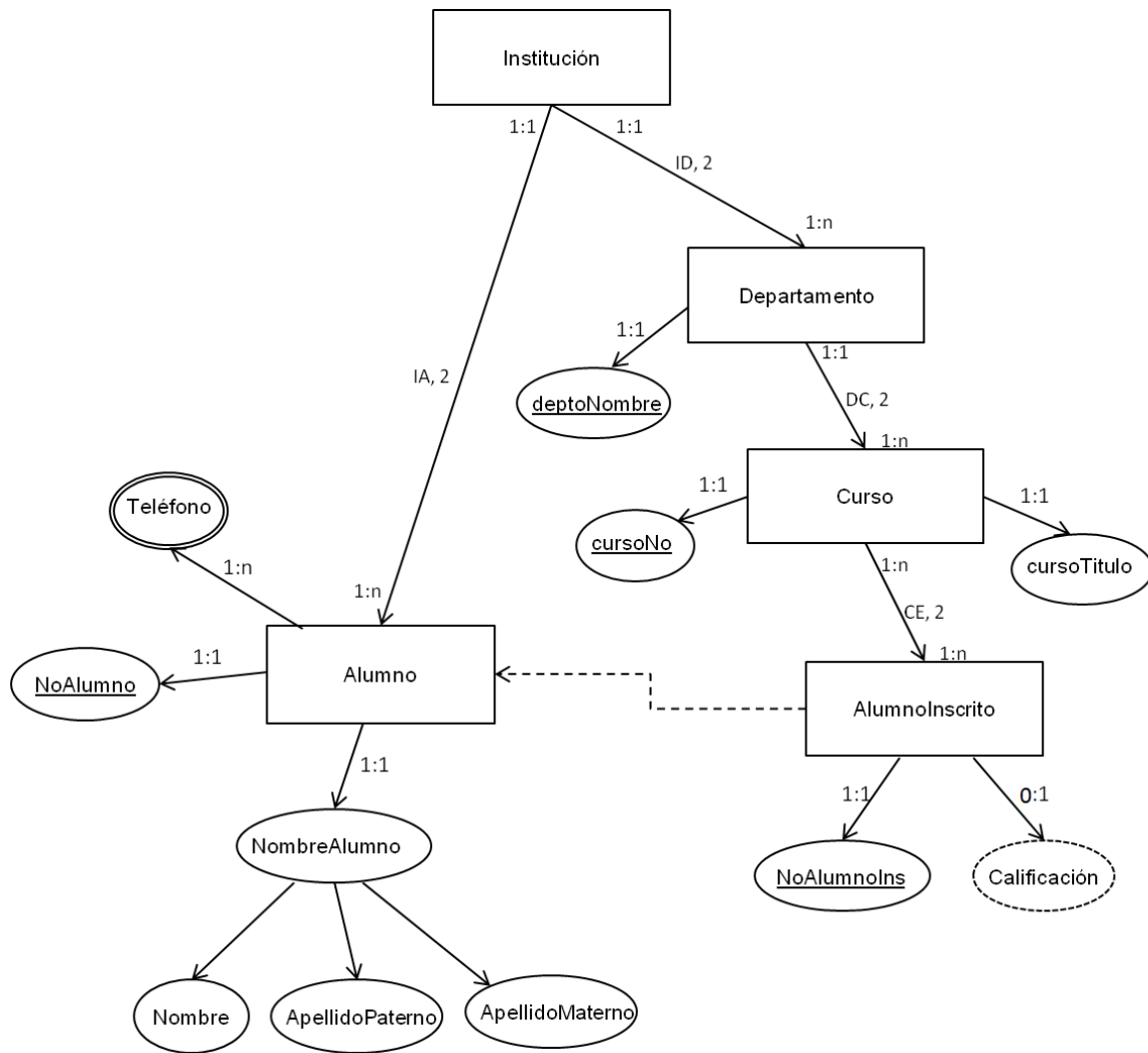


Figura 4.16 Representación gráfica de la referencia de la entidad Alumno Inscrito a la entidad Alumno

```

<Institucion>
<Alumnos>
  <Alumno NoAlumno=" A10">
    <NombreAlumno>
      <nombre> Juan</nombre>
      <apellidoPaterno> Morales</apellidoPaterno>
      <apellidoMaterno> Salas </ApellidoMaterno>
    </NombreAlumno>
    <Telefono>casa 55357689</Telefono>
    <Telefono>celular 55228699</Telefono>
  </Alumno>
  .
  .
  .
  
```

```

<AlumnoNoAlumno=" B200">
  <NombreAlumno>
    <nombre>Guadalupe</nombre>
    <apellidoPaterno> Mendoza </apellidoPaterno>
    <apellidoMaterno> Brindiz </ApellidoMaterno>
  </NombreAlumno>
  <Telefono> casa 44789065 </Telefono>
</Alumno>

<AlumnoNoAlumno=" A134">
  <NombreAlumno>
    <nombre> Felipe </nombre>
    <apellidoPaterno> Cornejo </apellidoPaterno>
    <apellidoMaterno> Salcedo </ApellidoMaterno>
  </NombreAlumno>
  <Telefono> celular 55789066 </Telefono>
</Alumno>
</Alumnos>


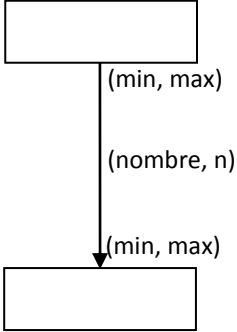
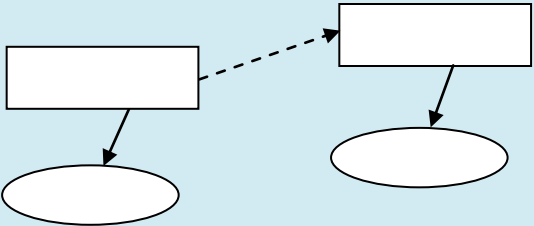




<Departamento deptoNombre="Ciencias de la Computación">
  <Curso cursoNo="101">
    <cursoTitulo> Introducción a la computación</cursoTitulo>
    <AlumnoInscrito NoAlumnoIns="A10">
      </AlumnoInscrito>
    .
    .
    .
  </Curso>
  <Curso cursoNo="105">
    <cursoTitulo> Lenguajes de programación</cursoTitulo>
    <AlumnoInscrito NoAlumnoIns="A10">
      <Calificacion> 9 </Calificacion>
    </AlumnoInscrito>
    <AlumnoInscrito NoAlumnoIns="A134">
      </AlumnoInscrito>
    .
    .
    .
  </Curso>
</Departamento>
</Institucion>

```

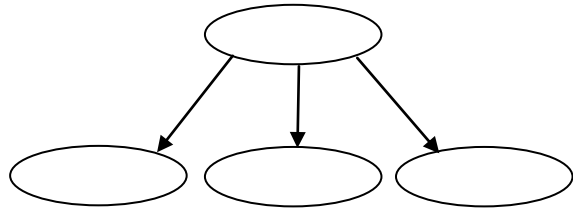
**Figura 4.17** Documento XML resultante del uso de Referencias

## 4.1 Notación Gráfica

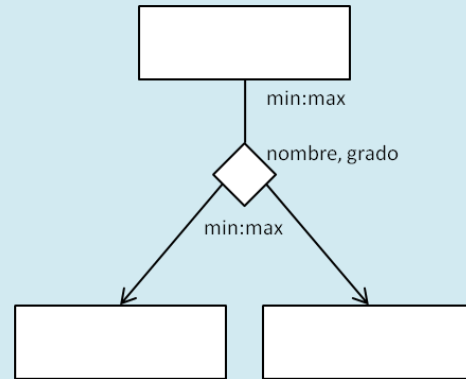
Un diagrama debe expresar gráficamente la estructura lógica de la Base de Datos. Los diagramas deben ser simples y claros, cualidades que permiten el amplio uso del modelo. El diagrama del modelo propuesto consta de los siguientes componentes:

Descripción	Notación Gráfica
<p><b>Conjuntos de entidades</b>, son representadas a través de un rectángulo, etiquetado con el nombre de la entidad.</p>	
<p><b>Relaciones</b>, son representadas a través de dos conjuntos de entidades unidas por una arista dirigida, donde la dirección va de la entidad padre a la entidad hijo, y una etiqueta con los valores del nombre y el grado de la relación (<b>nombre, n</b>) al centro de la flecha, las etiquetas de los extremos de la arista indican las restricciones de participación de las entidades padre e hijo respectivamente.</p>	
<p><b>Referencia</b>, es representada a través de dos entidades unidas mediante una flecha de línea punteada dirigida, que va desde la entidad que hace la referencia a la entidad referenciada, la referencia se realiza a través de dos atributos, cuyos valores deben coincidir en las instancias de las entidades que participan en la referencia.</p>	
<p><b>Atributos Monovalorados</b>. Son representados a través de una elipse.</p>	
<p><b>Atributos Multivalorados</b>. Son representados a través de una elipse de doble línea.</p>	
<p><b>Atributos Opcionales</b>. Son representados a través de una elipse de línea punteada.</p>	
<p><b>Atributos Llave</b>. Es un atributo elegido por el diseñador y es a su vez de tipo monovalorado, por lo que la representación gráfica es la misma, se identifica subrayando el nombre del atributo.</p>	

**Atributos Compuestos.** Son representados mediante elipses, colocando el nombre del atributo compuesto en la elipse principal, y unidos a esta las elipses que representan a los atributos que la componen.



**Disyunción de Entidades.** Se representa mediante un rombo, el cual une a la entidad padre con las entidades a elegibles, a través de una arista, se colocan las restricciones de participación, nombre y grado, al igual que en una relación.





## 4.2 Buenas prácticas aplicables al modelo

Un buen diseño es esencial para lograr los objetivos fijados para la base de datos. El diseño de una base de datos no es un proceso sencillo; habitualmente, la complejidad de la información y la cantidad de requisitos de los sistemas de información hacen que sea complicado. Por este motivo, se recomienda tomar en cuenta las siguientes buenas prácticas aplicables al modelo propuesto, para bases de datos Semi-estructurados:

- Identificar y entender los requisitos del problema que se va a modelar. Parece obvio, y debería serlo, pero es aquí donde comienza la mayoría de los problemas. Es fundamental leer los requisitos completamente y repetidamente hasta tenerlo todo 100%, claro y sin ambigüedades.
- Identificar las restricciones que necesitan ser capturadas en el modelo de datos, durante la fase del diseño; pueden ser plasmadas de forma escrita y posteriormente verificar que el modelo las refleje.
- Usar nombres descriptivos para las entidades del modelo.
- En la transformación del modelo a una DTD, un atributo monovalorado u opcional se puede traducir a un atributo o a un elemento, siendo el resultado de la transformación decisión del diseñador; a continuación se presentan algunos puntos que se deben tomar en cuenta al momento de la transformación:
  - Es complicado realizar cambios a los atributos, por ejemplo, si por alguna razón en un futuro se desea transformar un atributo monovalorado en multivalorado, un atributo no puede tener múltiples valores, mientras que un elemento sí.
  - Los atributos son más difíciles de manejar por un código de programa que un elemento.
  - Se recomienda utilizar atributos, sólo para proporcionar información que no es relevante.
- Tener en cuenta los siguientes puntos aplicables a algunos componentes del modelo:
  - Una relación con grado dos, relaciona a una entidad padre con una entidad hijo, siendo estas entidades distintas.
  - En una relación con grado  $n$ , donde  $n$  es mayor a dos, debe existir una relación con grado  $n-1$  que antecede a la entidad hijo de la relación.

- Una relación disyuntiva, relaciona a una entidad padre a dos o más entidades hijos diferentes.
- En una relación la mínima restricción de participación de una entidad padre, debe ser mayor a 0, desde que una entidad hijo debe estar relacionada al menos con una entidad padre.
- Un atributo compuesto se compone de dos o más atributos.
- Sólo puede existir un atributo llave por cada entidad.
- El número de valores de un atributo debe ser limitado por la restricción de participación del tipo de atributo (monovalorados, multivalorados, llave y opcionales).

## Capítulo 5. Reglas de transformación del modelo a una DTD (Definición de Tipo de Documento)

---

Una vez modelado un problema, aplicando el modelo propuesto se obtiene el diagrama correspondiente, se puede generar la Definición de Tipo de Documento (DTD) para XML, siguiendo las reglas de transformación de los componentes del diagrama, que a continuación se listan:

### Conjuntos de Entidades.

Para cada conjunto de entidades en el diagrama, generar definiciones de tipo de elemento.

**<!Element**nombre\_entidad>

Ó

**<!Element**nombre\_entidad(Lista de Subelementos)>

Ejemplo: En la Figura 5.1 se tienen los conjuntos de entidades: Departamento, Coordinador, Curso y Alumno, donde cada una se debe traducir a una definición de tipo de elemento, de la siguiente manera:

**<!ELEMENT Departamento>**

**<!ELEMENT Coordinador>**

**<!ELEMENT Curso>**

**<!ELEMENT Alumno>**

De acuerdo al diagrama del modelo, si un conjunto de entidades, tiene un subconjunto de entidades hijo, debe declararse un elemento, donde su contenido es de tipo: “otros elementos”, representado por: **<!Element (subelementList)>**, cuidando la secuencia de los elementos en la declaración, ya que la secuencia declarada será la secuencia en la que aparecerán las instancias de datos en el documento XML.

Ejemplo: En la figura 5.1, el conjunto de entidades Departamento tiene como entidades hijas: Curso y Coordinador, la entidad Curso tiene como entidad hija, a la entidad Alumno, las entidades padre se traducen a una definición de tipo de elemento: **<!Element (subelementList)>**, y las

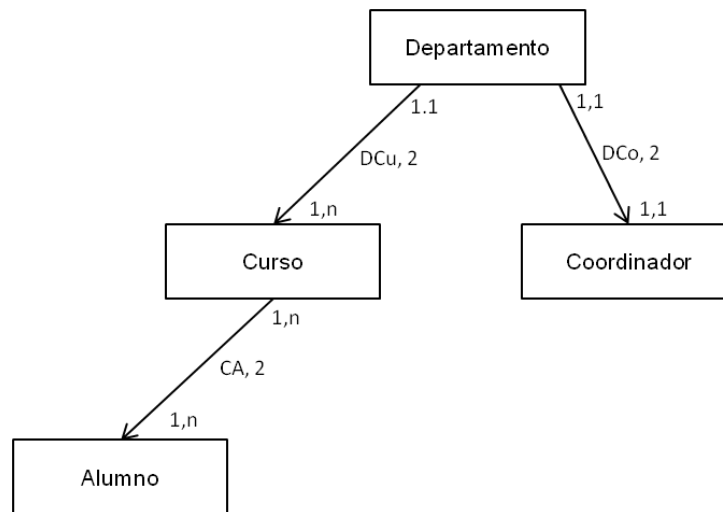
entidades que no sean padre, se traducen a una definición de tipo de elemento de la siguiente manera:

<!ELEMENT Departamento (Coordinador, Curso)>

<!ELEMENT Coordinador>

<!ELEMENT Curso (Alumno)>

<!ELEMENT Alumno>



**Figura 5.1** Ejemplo de entidades Departamento, Curso, Coordinador y Alumno

### Disyunción de Entidades.

En el caso de una disyunción, generar una definición de elemento para la entidad padre con listado de opciones, representado a través del símbolo “|”; y generar definiciones de tipo de elemento para cada subentidad hijo.

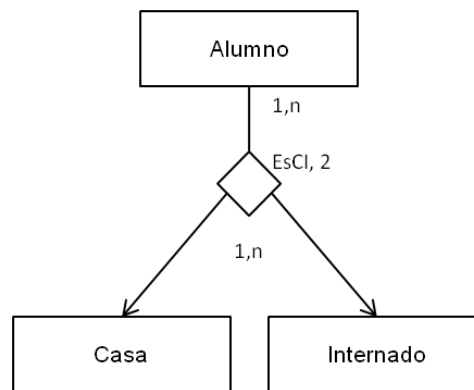
Ejemplo: En la figura 5.2, las entidades Casa e Internado son subentidades hijo de la entidad Alumno, representando una disyunción, en la cual un Alumno puede vivir en una Casa o en el Internado, pero no en ambos lugares, en este caso se debe generar una definición de tipo de elemento con un listado de opciones donde se colocarán los nombres de las entidades que

generan la disyunción, separadas con el símbolo “|”, esto para la entidad Alumno; y generar definiciones de tipo de elemento para la entidad Casa e Internado, de la siguiente manera:

**<!ELEMENT Alumno (Casa|Internado)>**

**<!ELEMENT Casa>**

**<!ELEMENT Internado>**



**Figura 5.2** Ejemplo de conjuntos de entidades Disyuntivas.

## Atributos

### Atributos Monovalorados

Para atributos Monovalorados generar una lista de definición de atributos para cada atributo. Los atributos monovalorados tienen una restricción de participación de 1,1, por lo que al momento de generar una definición de atributo se deberá especificar en la declaración la palabra **#REQUIRED**, que indica que el atributo es requerido, la declaración del atributo se hace de la siguiente manera:

**<!ATTLIST nombre\_elemento nombre\_atributo tipo #REQUIRED>**

Ejemplo: En el diagrama de la Figura 5.3, para el atributo monovalorado “sexo” de la entidad Alumno, se genera una lista de definición de atributos del elemento Alumno, que es la entidad a la que corresponde de la siguiente manera:

**<!ATTLIST Alumno sexo CDATA #REQUIRED >**

Se puede generar una definición de elemento para los atributos monovalorados, esto sólo en el caso de que el atributo no sea de tipo llave, y no se haya especificado un valor fijo o por default; es decisión del diseñador hacerlo, tomando en cuenta las recomendaciones de las buenas prácticas aplicables al modelo. En caso de que el diseñador opte por generar definiciones de tipo de elemento, se debe alterar a la entidad padre del atributo, para que lo incluya como subelemento.

En el siguiente ejemplo el atributo sexo de la entidad Alumno se transforma como un elemento, lo cual es posible, debido a que el atributo no es llave y no tiene asignado un valor fijo o por default, asignándolo como subelemento de la entidad Alumno, quedando de la siguiente manera:

<!ELEMENT Alumno (sexo)>

<!ELEMENT sexo>

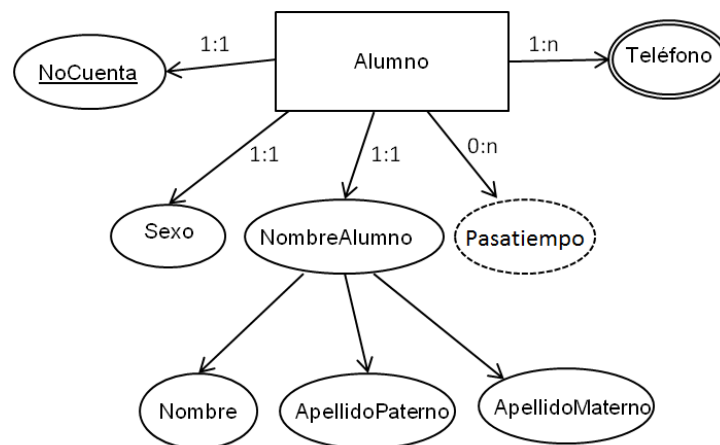


Figura 5.3 Diagrama de la entidad Alumno y sus atributos.

#### Atributos Monovalorados con valores Fijos.

Generar una lista de definición de atributos, para cada atributo, agregando la etiqueta #FIXED que indica que es un atributo de valor fijo, seguido del valor para el atributo:

<!ATTLIST nombre\_elemento nombre\_atributo tipo #FIXED "valor\_fijo">

### **Atributos Monovalorados con valores por Default.**

Generar una lista de definición de atributos para cada atributo, agregando el valor por default después del nombre del atributo:

**<!ATTLIST nombre\_elemento nombre\_atributo tipo "valor\_default">**

### **Atributos Multivalorados.**

Para los atributos Multivalorados, generar una definición de tipo elemento de la siguiente manera: **<!ELEMENT nombre\_elemento tipo>**, esto implica una alteración a la entidad padre, por lo que se debe realizar una modificación para que el atributo quede como una subentidad del mismo; en lo que respecta a la restricción de participación, este tipo de atributos tiene una restricción de participación 1,n, por lo que se debe indicar el símbolo "+" en la declaración de la entidad padre, expresando que la entidad padre puede tener de uno a un número indefinido de subentidades hijo.

Ejemplo: En el diagrama de la Figura 5.3, para el atributo multivalorado Teléfono de la entidad Alumno, se realiza una definición de elemento de Teléfono, así como la modificación de la declaración de la entidad padre Alumno, quedando Teléfono como una subentidad e indicando la restricción de participación:

**<!ELEMENT Alumno (Teléfono+)>**

**<!ELEMENT Teléfono>**

### **Atributos Compuestos**

Para los atributos compuestos generar una definición de elemento del atributo principal de la siguiente manera: **<!ELEMENT nombre\_elemento (lista de subelementos)>**; colocar a los atributos que lo componen en la lista de subelementos; modificar al elemento padre el cual es la entidad a la que corresponde el atributo compuesto, agregando en su lista de subelementos al atributo principal, indicando la restricción de participación, la cual se debe indicar con alguno de los símbolos siguientes: +(indica que el elemento aparece 1:n veces), \* (0:n, 0 o más veces), ?(0:1, 0 o una vez) y en el caso de que la ocurrencia sea 1:1, no se indica nada en la declaración; por

último se deben generar definiciones de tipo elemento para los elementos que componen al atributo; la sintaxis de declaración es la siguiente:

```
<!ELEMENT nombre_elemento_compuesto (elemento1, elemento2, elemento3...)>
<!ELEMENT elemento1>
<!ELEMENT elemento2>
<!ELEMENT elemento3>
.
.
```

Ejemplo: En el diagrama de la Figura 5.3, el compuesto es el NombreAlumno se compone de los atributos: nombre, apellidoPaterno y apellidoMaterno; para la transformación se genera una definición de elemento con lista de subelementos para el atributo NombreAlumno, asignando en la lista, el nombre de los atributos que lo componen; posteriormente se agrega NombreAlumno, a la lista de subelementos generada para el atributo Alumno; tomando en cuenta que la restricción de participación de todos los atributos es 1:1, no se indica ningún símbolo en las listas de subelementos correspondientes; por último se generan definiciones de tipo elemento para los atributos: nombre, apellidoPaterno y apellidoMaterno, quedando la declaración de la siguiente manera:

```
<!ELEMENT Alumno (NombreAlumno)>
<!ELEMENT NombreAlumno(nombre, apellidoPaterno, apellidoMaterno)>
<!ELEMENT nombre>
<!ELEMENT apellidoPaterno>
<!ELEMENT apellidoMaterno>
```

### Atributos Opcionales

Para los atributos opcionales, en el caso donde la restricción de participación sea **0,n**, se debe generar una definición de elemento del atributo con la siguiente sintaxis: **<!ELEMENT nombre\_elemento>**, agregando al nuevo elemento en la lista de subelementos de la entidad a la que corresponda del atributo, indicar también el símbolo de ocurrencia del elemento como opcional, el cual corresponde a **"\*"**.



Ejemplo: En el diagrama de la Figura 5.3, para el atributo opcional Pasatiempo de la entidad Alumno, se genera una definición de elemento, agregando a este, como subelemento de la entidad Alumno de la siguiente manera:

```
<!ELEMENT Alumno (Pasatiempo*)>
<!ELEMENT Pasatiempo>
```

Para el caso donde la restricción de participación sea 0,1, generar una lista de definición de atributos con la etiqueta **#IMPLIED**, que indica que el atributo es opcional siguiendo la sintaxis: **<!ATTLIST nombre\_elemento nombre\_atributo tipo #IMPLIED >**; “nombre\_elemento” es el nombre de la entidad a la que corresponde el atributo.

Ejemplo: En el diagrama de la Figura 5.3, para el atributo opcional Pasatiempo de la entidad Alumno, se genera una lista de definición de atributos correspondiente a la entidad Alumno:

```
<!ATTLIST Alumno Pasatiempo CDATA #IMPLIED>
```

Nota: Es decisión del diseñador, de acuerdo a las buenas prácticas aplicables al modelo, generar una definición de elemento, si es este el caso, se debe agregar el elemento generado a la lista de subelementos correspondientes a la entidad a la que pertenece el atributo, indicando el símbolo de ocurrencia correspondiente, el cual es: “?”. Si se opta por generar una definición de tipo elemento para el atributo Pasatiempo con restricción de participación 0:1, se debe agregar el elemento resultante en la lista de subelementos de la entidad Alumno, quedando la declaración de la siguiente manera:

```
<!ELEMENT Alumno (Pasatiempo?)>
<!ELEMENT Pasatiempo>
```

### **Atributos Llave**

Para los atributos llave se debe generar una lista de definición de atributos, agregando la etiqueta **ID** y **#REQUIRED**, que indican que son valores únicos y requeridos respectivamente; la sintaxis de declaración es la siguiente:

```
<!ATTLIST nombre_elemento nombre_atributo ID #REQUIRED >
```

Ejemplo: En el diagrama de la Figura 5.3, para el atributo llave NoCuenta de la entidad Alumno, se genera una lista de definición de atributos de la siguiente manera:

**<!ATTLIST Alumno NoCuenta ID #REQUIRED >**

## Relaciones

Un documento XML consta de elementos anidados, una relación en XML es un anidamiento de elementos. Anteriormente, se indicó la forma de traducir las entidades del diagrama a elementos, pero en un modelo de datos semi-estructurados las entidades no se encuentran sueltas, siempre se encontrarán anidadas en forma de relaciones. Las relaciones en el modelo, están representadas mediante una flecha dirigida de padre a hijo, y una etiqueta que indica el grado de la relación (binaria, ternaria, etc.), la restricción de participación de la entidad padre e hijo se indican a los extremos de la relación.

Los indicadores de frecuencia: “?”, “\*”, “+”, son utilizados para representar las restricciones de participación de las entidades en la relación.

- + Incluye una o más ocurrencias de ese elemento, lo que significa, que debe existir al menos una instancia de la entidad.
- \* Incluye 0 o más ocurrencias del elemento, lo que significa que pueden no existir instancias de la entidad.
- ? El elemento es opcional, es decir puede existir sólo una instancia de la entidad o ninguna.

Cuando la restricción de participación en 1:1, no se indica ningún símbolo en la declaración.

Reglas de transformación de los indicadores de frecuencia:

La entidad raíz del diagrama tiene una restricción de participación: 1:1, y puede o no indicarse en la etiqueta del diagrama, en este caso se genera una definición de elemento y no se tiene que indicar la restricción de participación en la declaración de la DTD.

En una relación, se realiza la definición de elemento para la entidad padre con una lista de subelementos, donde se colocan los nombres de las entidades hijo que le corresponden y el símbolo correspondiente a la restricción de participación de la entidad hijo, en caso de que sea **1:n**, donde **n**, es un número indeterminado de valores, se coloca el símbolo: “+”; si la restricción de participación es: **0:n**, se colocará el símbolo: “\*”; o en el caso de que la restricción de participación sea: **0:1**, se colocará el símbolo: “?”; si la restricción de participación es: **1:1**, no se coloca ningún símbolo.

**Relación Binaria:** En una relación binaria participan dos entidades, la restricción de participación del padre puede ser: **1:1** o **1:n**, ambas se transforman de la misma manera a una DTD. Se genera una definición de elemento para la entidad padre con lista de subelementos, donde se colocará el nombre y la restricción de participación de la entidad hijo, posteriormente se genera una definición de tipo de elemento para la entidad hijo. La sintaxis de declaración es la siguiente:

```
<!ELEMENT nombre_elemento_padre (nombre_elemento_hijo(+|*|?))>  
<!ELEMENT nombre_elemento_hijo>
```

Nota: En el caso de que la restricción de participación de la entidad padre sea: **1:n**, no se toma en cuenta al momento de la transformación al DTD; sólo se considera de carácter informativo, para tener en cuenta que la información de algunas instancias de la entidad hijo, puede encontrarse duplicada en varias instancias de la entidad padre.

Ejemplo: En el diagrama de la Figura 5.4, existen dos relaciones binarias, una relación de nombre **DCu** entre las entidades Departamento y Curso, y la segunda de nombre **DCo** entre las entidades Departamento y Coordinador, para ejemplificar la traducción del modelo al DTD, para la relación binaria DCu, se genera una definición de tipo elemento, para la entidad padre Departamento, poniendo en la lista de subelementos el nombre de la entidad hijo Curso, así como el símbolo de la restricción de participación: “+” correspondiente a: **1:n**; posteriormente, se genera una definición de tipo elemento para la entidad hijo Curso, quedando de la siguiente manera:

```
<!ELEMENT Departamento (Curso+)>  
<!ELEMENT Curso>
```

A continuación, se muestra la transformación de las dos relaciones binarias del diagrama de la Figura 5.4 a una DTD.

<!ELEMENT Departamento (Coordinador+, Curso+)>

<!ELEMENT Coordinador>

<!ELEMENT Curso>

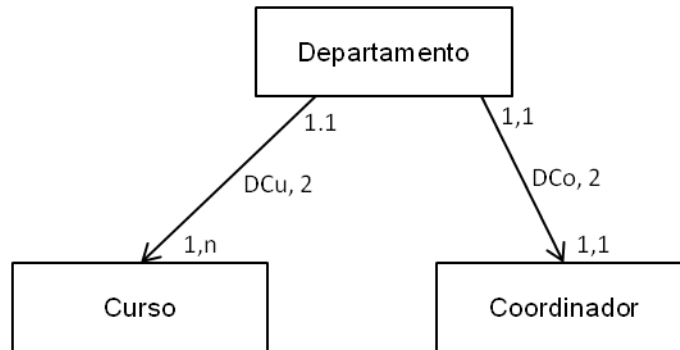


Figura 5.4 Ejemplo de Relaciones Binarias

**Relación Ternaria:** Una relación ternaria es una relación binaria entre dos entidades, y esta relación binaria se relaciona con otra entidad, para la transformación a una DTD, se generan definiciones de elementos para cada entidad participante en la relación, creando un anidamiento de forma jerárquica, desde la entidad padre de la relación, hasta la última entidad hija, indicando las restricciones de participación del anidamiento de la siguiente manera:

<!ELEMENT nombre\_elemento\_padre (nombre\_elemento\_hijo1(+|\*|?))>

<!ELEMENT nombre\_elemento\_hijo1(nombre\_elemento\_hijo2(+|\*|?))>

<!ELEMENT nombre\_elemento\_hijo2>

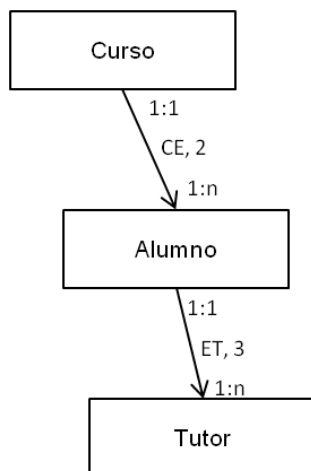
Ejemplo: En el siguiente diagrama existe una relación ternaria entre las entidades Curso, Alumno y Tutor, en primer lugar se identifica a la relación binaria entre la entidad Curso y Alumno, posteriormente esta relación se relaciona con la entidad Tutor; se generan definiciones de tipo elemento para cada una de las entidades en la relación binaria, se coloca como subentidad a la entidad hijo Alumno y su restricción de participación en la entidad padre Curso, después en la declaración de la entidad Alumno, se coloca como subentidad a la entidad Hijo Autor con su

restricción de participación; quedando la declaración con el anidamiento correspondiente de la siguiente manera:

**<!ELEMENT Curso (Alumno+)>**

**<!ELEMENT Alumno(Tutor)>**

**<!ELEMENT Tutor>**



**Figura 5.5** Ejemplo de relación Ternaria.

## Referencias

Las referencias se realizan mediante un atributo de tipo ID, que identifica de manera única a un elemento y un atributo de tipo IDREF que apunta a un atributo de tipo ID (estos valores deben tener correspondencia en ambas entidades, ya que a través de ellos se realiza la referencia), el parser valida que cada atributo ID referenciado por el atributo IDREF, exista en el documento XML.

Para las referencias encontradas en el diagrama, generar definiciones de tipo elemento para las entidades que participan en la referencia; generar una lista de definición de atributos de tipo ID, para el atributo correspondiente a la entidad referenciada, y una lista de definición de atributos de tipo IDREF, del atributo correspondiente a la entidad que hace la referencia; la sintaxis para la declaración de referencias es la siguiente:

**<!ELEMENT entidad\_referenciada>**

**<!ATTLIST entidad\_referenciada nombre\_atributo ID #REQUIRED>**

**<!ELEMENT entidad\_referencia>**

**<!ATTLIST entidad\_referencia nombre\_atributo IDREF #REQUIRED>**

Ejemplo: En el diagrama de la Figura 5.6, en la referencia entre las entidades AlumnoInscrito y Alumno, la entidad que hace la referencia, es la entidad AlumnoInscrito y la entidad referenciada, es la entidad Alumno, la entidad Alumno contiene a todos los alumnos de la institución y la entidad AlumnoInscrito, sólo hace referencia a los alumnos de la entidad Alumno que se encuentren inscritos en un Curso en particular, se generan definiciones de elemento para cada una de las entidades; se genera una lista de definición de atributos tipo ID, para el atributo NoAlumno correspondiente a la entidad Alumno; así como una lista de definición de atributos de tipo IDREF, para el atributo NoAlumnoIns correspondiente a la entidad AlumnoInscrito, quedando la declaración de la siguiente manera:

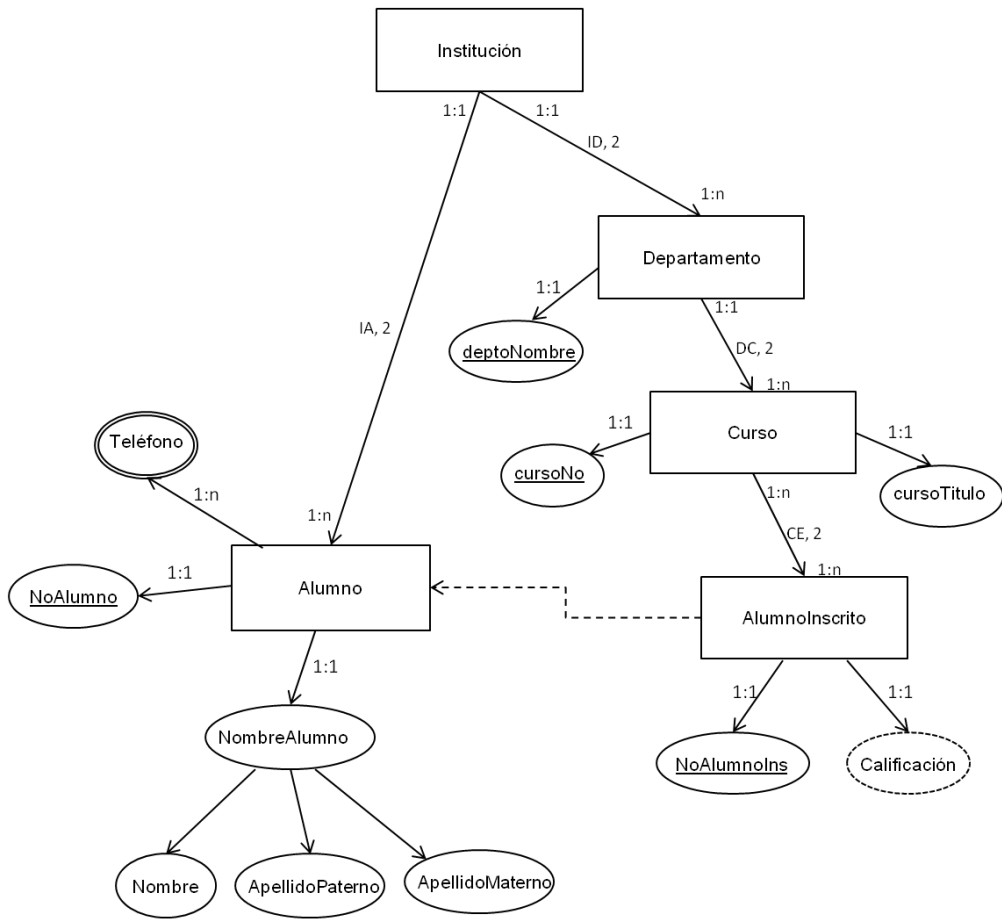
**<!ELEMENT Alumno (...)>**

**<!ATTLIST Alumno NoAlumno ID #REQUIRED>**

**<!ELEMENT Curso(AlumnoInscrito)>**

**<!ELEMENT AlumnoInscrito>**

**<!ATTLIST AlumnoInscrito NoAlumnoIns IDREF #REQUIRED>**



**Figura 5.6** Ejemplo de uso de Referencias.

## Capítulo 6. Modelado de sistema: Librería

---

### 6.1 Definición del Problema

Se desea definir una base de datos en XML para una librería, que permita gestionar la información referente a los libros que tiene a la venta; por cada libro se debe almacenar la siguiente información:

- ✓ Título del libro, es un dato obligatorio y dos libros o más pueden tener el mismo título.
- ✓ Autor del libro, cada uno de los libros debe tener al menos un autor.
- ✓ ISBN (Es un código internacional que identifica unívocamente el libro) del libro, todos los libros deberán contar con ISBN distinto.
- ✓ Editorial.
- ✓ Año de edición y año de publicación.
- ✓ Precio de adquisición y de venta al público, del libro.
- ✓ Género: (terror, acción,...) Se ha de clasificar dentro de un único género, como campo obligatorio.
- ✓ Número de páginas, el cual es un dato opcional del libro.
- ✓ Datos de ubicación del libro, Número de la estantería donde se encuentra el libro y el pasillo dentro de la librería.
- ✓ Número de ejemplares.
- ✓ Observaciones, donde puede figurar un resumen.
- ✓ Como dato opcional debe poder almacenarse la imagen de la portada del libro.

La librería dispone de un archivo de autores, en el que se almacena información referente al nombre, primer apellido, datos de nacimiento: fecha, nacionalidad y ciudad de nacimiento, Instituciones donde cursó sus estudios, fotografía y un campo de observaciones.

De las editoriales se almacenará: nombre, teléfono, e-mail, persona de contacto, dirección, un campo de observaciones y un código único para diferenciarla.



## 6.2 Desarrollo del Diagrama

Como primer paso, se deben identificar los requisitos del problema que se va a modelar, por lo que se identificaron los siguientes elementos:

**Conjunto de Entidades.** Las entidades identificadas en la definición del problema son: librería, libro, autor y editorial. Se ilustran en la Figura 6.1.



**Figura 6.1.** Representación gráfica de las entidades identificadas en el problema

### Atributos

Para cada conjunto de entidades, se identifican sus respectivos atributos; cada diseñador puede modelar el problema de una manera distinta, de acuerdo a su estilo de diseño. La entidad Librería se identifica como la entidad raíz en el modelo, la cual no contiene ningún atributo. A continuación, se describen los atributos identificados de las entidades, en el diseño:

Para el conjunto de entidades Libro se identifican los atributos monovalorados: título del libro, observaciones, año de publicación, año de edición, género del libro, número de ejemplares y editorial; los atributos precio de adquisición y precio de venta al público se van a integrar en un atributo compuesto, llamado precio del libro; el número de estantería y número de pasillo donde se ubica el libro, se modelan mediante un atributo compuesto de nombre: ubicación del libro; los atributos portada y número de páginas, se modelan como atributos opcionales, ya que no es necesario que para todas las instancias del conjunto de entidades aparezcan estos datos; el atributo ISBN del libro, es el atributo elegido como llave, ya que permite asegurar que en un documento, el ISBN de cada libro es único. La Figura 6.2 ilustra la representación gráfica del conjunto de entidades Libro y sus respectivos atributos. La mayoría de los atributos tienen una

restricción de participación 1:1, ya que son datos obligatorios y sólo puede existir un dato para cada atributo; los atributos portada y número de páginas, tienen una restricción de participación 0:1, ya que son atributos opcionales.

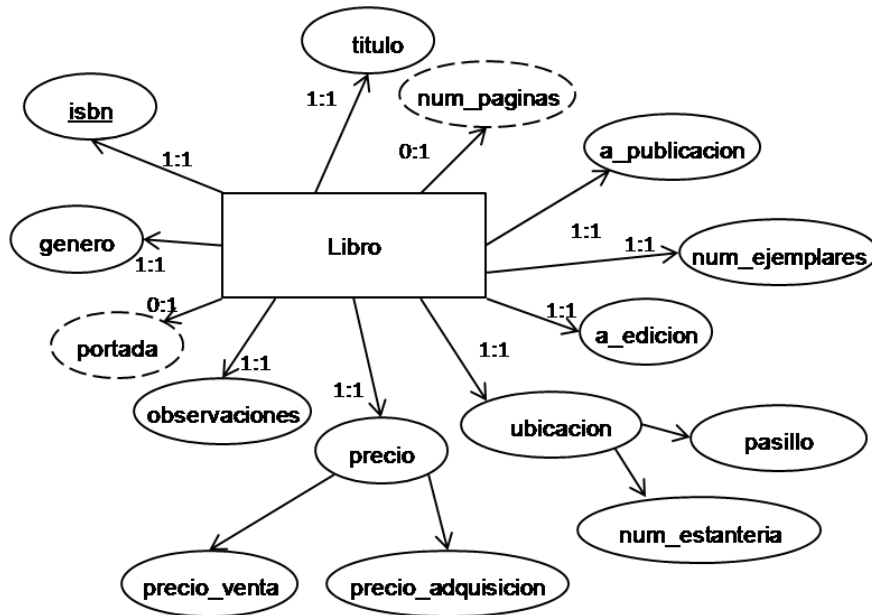
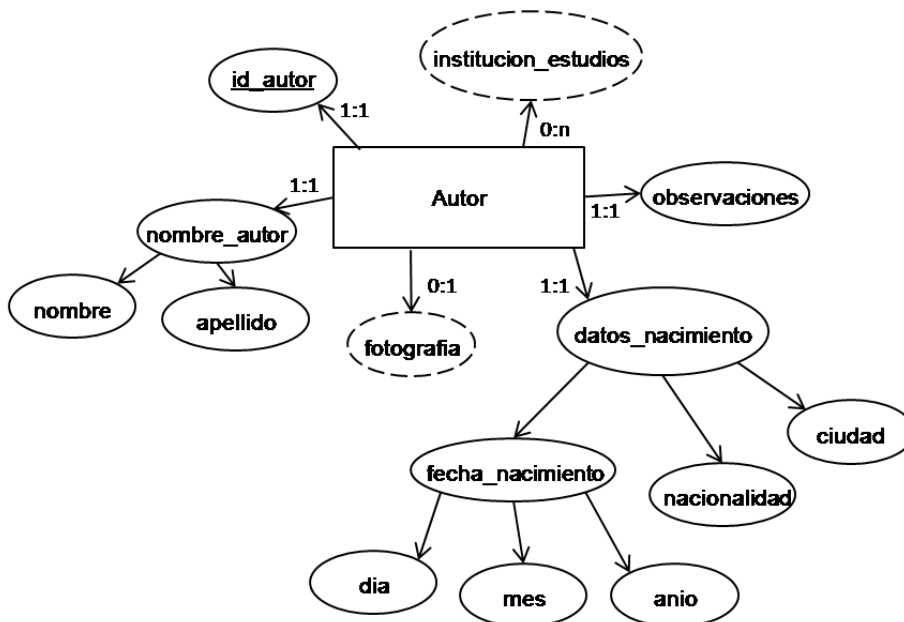


Figura 6.2. Representación gráfica del conjunto de entidades Libro y sus respectivos atributos

En el conjunto de entidades Autor, la propiedad observaciones se identifica como un atributo monovalorado; se identifica el atributo compuesto: nombre del autor, el cual se compone de los atributos: nombre y apellido; el atributo datos de nacimiento es un atributo compuesto, al constar de los atributos: fecha de nacimiento, nacionalidad y ciudad de nacimiento, a su vez el atributo fecha de nacimiento es también un atributo compuesto, que se compone de los atributos: día, mes y año; se identifican los atributos opcionales: fotografía e institución(es) donde cursó los estudios el autor; se coloca un atributo identificador de cada autor en el documento, el cual se denomina: id\_autor. El atributo monovalorado observaciones, y los atributos compuestos: nombre del autor y datos de nacimiento, tienen una restricción de participación 1:1, ya que son datos obligatorios; el atributo opcional: fotografía, tiene una restricción de participación: 0:1, debido a que puede no aparecer en el documento y si existe puede sólo existir una fotografía por cada autor; el atributo opcional: institución donde cursó sus estudios, tiene restricción de participación

0:n, ya que puede no existir información o registrar varias instituciones. El conjunto de entidades Autor y sus respectivos atributos se representan en la Figura 6.3.



**Figura 6.3.** Representación gráfica del conjunto de entidades Autor y sus respectivos atributos

En el conjunto de entidades Editorial, se identifican los atributos monovalorados: nombre y observaciones, estos atributos tienen restricción de participación 1:1, ya que son datos obligatorios y sólo puede existir un dato por cada atributo; el atributo e-mail y teléfono son atributos multivalorados y su restricción de participación es 1:n, lo que significa que al menos se debe contar con un número de teléfono y un e-mail registrado; el atributo dirección y persona de contacto, se identificaron como atributos compuestos con restricción de participación 1:1, el atributo persona de contacto consta de los atributos: nombre y apellido; el atributo dirección se compone de los atributos: calle, número, colonia, estado y código postal; para identificar a cada una de las editoriales en un documento, se define al atributo llave: id\_editorial. El conjunto de entidades Editorial y sus respectivos atributos se representan en la Figura 6.4.

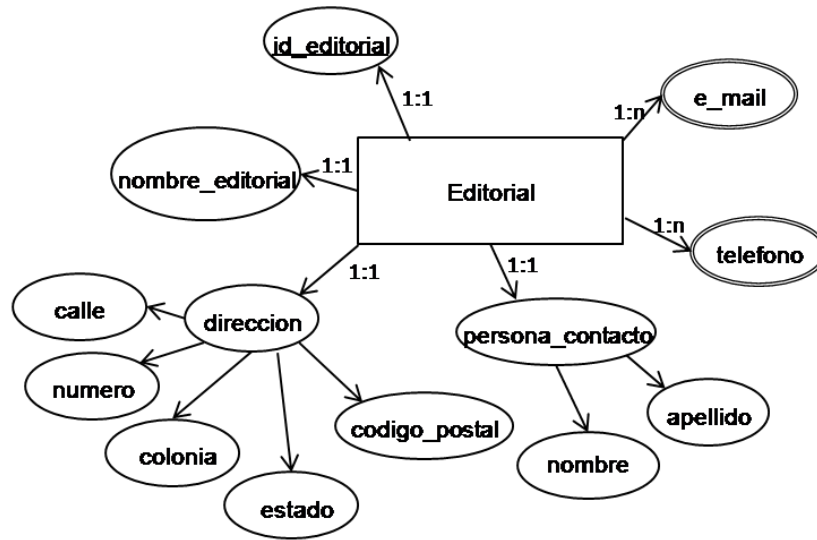


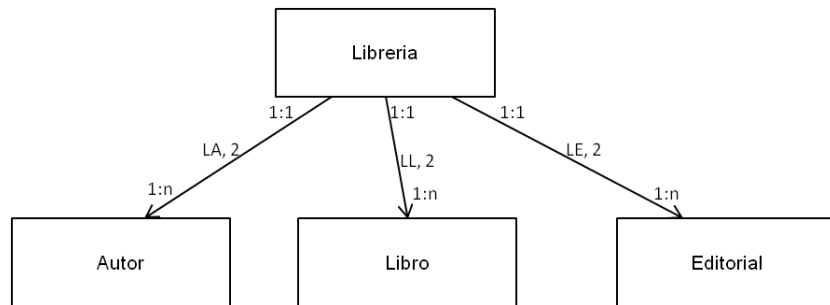
Figura 6.4. Representación gráfica del conjunto de entidades Editorial y sus respectivos atributos

### Relaciones

Como primera impresión se identifican dos relaciones binarias, una entre las entidades libro y editorial, y otra entre las entidades libro y autor, sin embargo, si se modela de esta manera se permitirá que los documentos puedan introducir información redundante, este efecto se presentaría, debido a que un libro puede tener uno o más autores, por lo que, los datos de un autor podrían estar replicados en distintas instancias de la entidad libro, lo mismo ocurre con los datos de una editorial de un libro.

Para evitar la redundancia las relaciones antes mencionadas, las relaciones se modelan usando referencias, lo que implica que en el modelo resultante existan tres relaciones binarias: una relación entre la entidad raíz librería y la entidad autor (LA, 2), otra entre librería y libro (LL,2), y una última entre librería y editorial (LE, 2), en la representación gráfica de cada relación se debe colocar el nombre y grado de las relaciones, en este caso el nombre que se les da a las relaciones, son las iniciales de los nombre de las entidades que conforman a la relación, y debido a que la relación es binaria se debe indicar el grado con un número 2; la restricción de participación de la entidad padre en todas las relaciones es: 1:1, ya que las instancias de las entidades hijo, sólo pueden pertenecer a una librería; la restricción de participación de las entidades hijo en cada una de las relaciones es: 1:n, esto, debido a que en la librería debe de existir al menos una instancia de

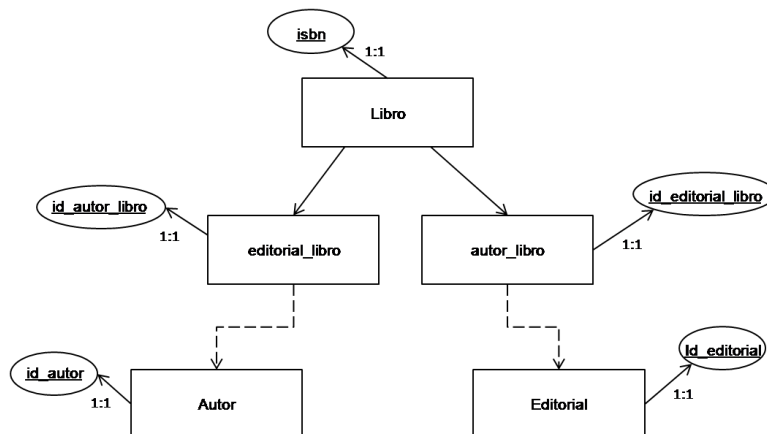
la entidad libro, y por lo tanto, una instancia de las entidades Autor y Editorial. La figura 6.5 ilustra la representación gráfica de las relaciones descritas.



**Figura 6.5.** Representación gráfica de las relaciones identificadas en la definición del problema

## Referencias

Para evitar la redundancia de información y alguna anomalía en la actualización de datos, las relaciones del problema entre libro y autor, y, entre libro y editorial, se modelan a través de referencias, una referencia es utilizada para introducir la información de los autores del libro, y la otra para introducir la información de la editorial. Para el modelado de las referencias en el diagrama, se generan dos entidades con los atributos necesarios. Los atributos deben tener valores que coincidan en cada una de las instancias, para realizar la referencia. La figura 6.5 ilustra el modelado de las referencias en el problema.



**Figura 6.5.** Representación gráfica de las referencias.

## 6.2.1 Modelo de Base de datos: Librería

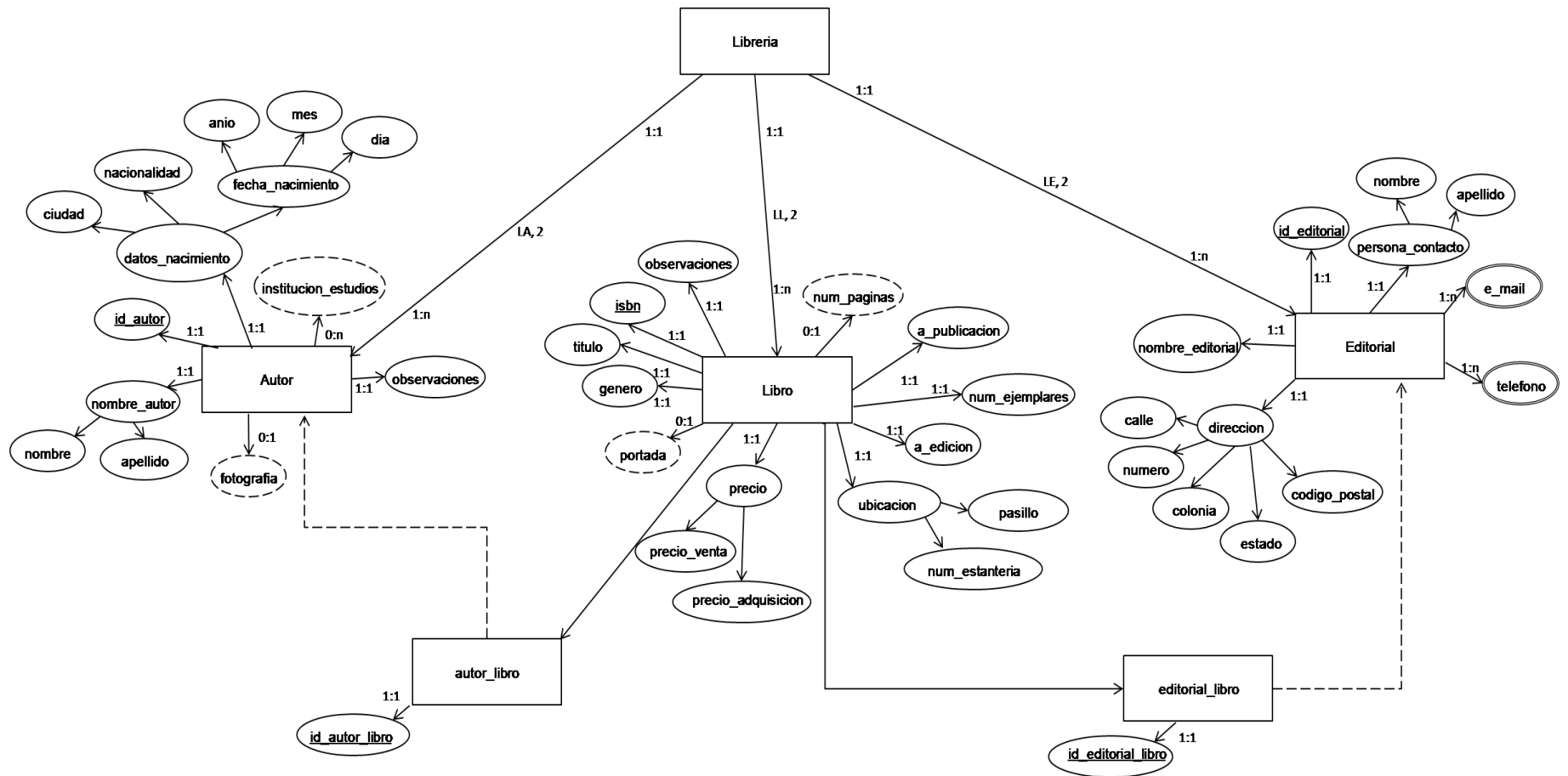


Figura 6.6. Diagrama Final del modelado del problema

### 6.3 Traducción del modelo a la DTD (Definición de Tipo de Documento).

Para generar la DTD correspondiente al modelo, se aplican las reglas de transformación, que se dieron a conocer en el capítulo 5 del presente trabajo.

Para cada conjunto de entidades identificado en el diagrama resultante, se genera una definición de tipo elemento en la DTD quedando de la siguiente manera:

```
<!ELEMENT Libreria>  
<!ELEMENT Libro>  
<!ELEMENT Editorial>  
<!ELEMENT Autor>
```

#### Relaciones

De acuerdo al diagrama resultante, se identifican tres relaciones binarias, donde la entidad el padre de la relación en todos los casos, es el conjunto de entidades Libreria, para este conjunto de entidades, se generó una definición de elemento, a la cual se le deben de agregar sus entidades hijo con su respectiva restricción de participación: 1:n, la cual se indica a través del símbolo "+", la transformación de las relaciones del diagrama quedan como sigue:

```
<!ELEMENT Librería (Libro+, Autor+, Editorial+)>
```

Cada conjunto de entidades tiene una serie de atributos, los cuales deben ser transformados a componentes de la DTD, por lo cual se realiza la transformación de sus atributos. Con el objetivo de tener presentes los atributos, su tipo y su restricción de participación, se listan los conjuntos de entidades y la información correspondiente a cada uno, para así visualizar fácilmente la transformación.

**Conjunto de Entidades:** Libro

**Atributos:**

- isbn (llave | 1:1)

- observaciones (monovalorado | 1:1)
- titulo (monovalorado | 1:1)
- genero (monovalorado | 1:1)
- a\_edicion (monovalorado | 1:1)
- a\_publicacion (monovalorado | 1:1)
- num\_ejemplares (monovalorado | 1:1)
- portada (opcional | 0:1)
- num\_paginas (opcional | 0:1)
- precio (compuesto por los atributos: precio\_venta y precio\_adquisicion | 1:1)
- ubicacion (compuesto por los atributos: pasillo y num\_estanteria | 1:1)

En el caso de los atributos monovalorados, debido a que son datos que agregan valor a la información, se generan definiciones de elementos para cada uno, integrándolos en la entidad correspondiente quedando de la siguiente manera:

```

<!ELEMENT Libro (titulo, genero, a_edicion, a_publicacion, num_ejemplares,
observaciones)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT genero (#PCDATA)>
<!ELEMENT a_edicion (#PCDATA)>
<!ELEMENT a_publicacion (#PCDATA)>
<!ELEMENT num_ejemplares (#PCDATA)>
<!ELEMENT observaciones (#PCDATA)>

```

Para el caso de los atributos opcionales, se genera una lista de atributos, indicando su tipo y la palabra reservada IMPLIED, que indica que el atributo puede o no existir en la instancia de datos, quedando de la siguiente manera:

```

<ATTLIST Libro num_pagina CDATA #IMPLIED
portada CDATA #IMPLIED >

```

Para los atributos compuestos: precio y ubicación, se genera una definición de elemento para el atributo raíz y definiciones de elemento para los atributos que lo componen, incluyendo estas declaraciones en el elemento raíz, quedando de la siguiente manera:

```

<!ELEMENT ubicacion (pasillo, num_estanteria)>
<!ELEMENT pasillo (#PCDATA)>
<!ELEMENT num_estanteria(#PCDATA)>

```



```
<!ELEMENT precio (precio_adquisicion, precio_venta)>
<!ELEMENT precio_adquisicion (#PCDATA)>
<!ELEMENT precio_venta (#PCDATA)>
```

El atributo llave que identifica a cada una de las instancias del conjunto de entidades Libro, debe incluirse en la definición de lista de atributos de la entidad, con las palabras reservadas ID y REQUIRED, quedando la definición de atributos de la siguiente manera:

```
<!ATTLIST Libro isbn ID #REQUIRED
                num_pagina CDATA #IMPLIED
                portada CDATA #IMPLIED >
```

Finalmente, a la definición de elemento correspondiente a la entidad Libro, se le deben agregar las definiciones de elementos y atributos generados quedando como sigue:

```
<!ELEMENT Libro (titulo, genero, a_edicion, a_publicacion, num_ejemplares,
                 observaciones, ubicacion, precio)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT genero (#PCDATA)>
<!ELEMENT a_edicion (#PCDATA)>
<!ELEMENT a_publicacion (#PCDATA)>
<!ELEMENT num_ejemplares (#PCDATA)>
<!ELEMENT observaciones (#PCDATA)>
<!ELEMENT ubicacion(pasillo, num_estanteria)>
<!ELEMENT pasillo (#PCDATA)>
<!ELEMENT num_estanteria (#PCDATA)>

<!ELEMENT precio(precio_adquisicion, precio_venta)>
<!ELEMENT precio_adquisicion (#PCDATA)>
<!ELEMENT precio_venta (#PCDATA)>

<!ATTLIST Libro isbn ID #REQUIRED
                num_pagina CDATA #IMPLIED
                portada CDATA #IMPLIED >
```

**Conjunto de Entidades:** Autor

**Atributos:**

- id\_autor (llave | 1:1)
- observaciones (monovalorado | 1:1)

- fotografia (opcional | 0:1)
- institucion\_estudios (opcional | 0:n)
- nombre\_autor (compuesto por los atributos: nombre y apellido | 1:1)
- datos\_nacimiento (compuesto por los atributos: nacionalidad, ciudad, y fecha\_nacimiento, el cual también es compuesto, por los atributos: dia, mes y anio | 1:1)

Para los atributos monovalorados, se generan definiciones de elemento y se incluyen en la definición de tipo de elemento como sigue:

```
<!ELEMENT Autor (observaciones)>
<!ELEMENT observaciones (#PCDATA)>
```

Para la transformación de los atributos compuestos, se genera una definición de elemento para el atributo raíz, y definiciones de elemento para los atributos que lo componen, esta misma regla se sigue para los atributos compuestos anidados, posteriormente se integran en la definición de elemento a la que corresponden, quedando de la siguiente manera:

```
<!ELEMENT nombre_autor(nombre, apellido)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>

<!ELEMENT datos_nacimiento(nacionalidad, ciudad, fecha_nacimiento)>
<!ELEMENT nacionalidad (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT fecha_nacimiento (dia, mes, anio)>
<!ELEMENT dia (#PCDATA)>
<!ELEMENT mes (#PCDATA)>
<!ELEMENT anio (#PCDATA)>
```

En el caso de los atributos opcionales, para el atributo fotografía, que tiene restricción de participación: 1:0, se genera una definición de elemento, indicando el tipo y la palabra reservada IMPLIED; para el atributo opcional: institucion\_estudios, que tiene una restricción de participación: 0:n, se genera una definición de elemento y se incluye en la sublista de elementos de la definición del conjunto de entidades Autor, agregando el símbolo de ocurrencia: "\*", quedando de la siguiente manera:

```
<!ELEMENT Autor (institucion_estudios*)>
<!ELEMENT institucion_estudios (#PCDATA)>
```

**<!ATTLIST Autor fotografia CDATA #IMPLIED >**

Para la transformación del atributo llave id\_autor, se agrega el atributo a la definición de lista de atributos existente, quedando la definición de la siguiente manera:

**<!ATTLIST Autor fotografia CDATA #IMPLIED  
id\_autor ID #REQUIRED >**

Integrando las transformaciones del conjunto de entidades Autor y sus atributos la definición de la DTD queda de la siguiente manera:

**<!ELEMENT Autor (nombre\_autor, datos\_nacimiento, observaciones,  
institucion\_estudios\*)>**

**<!ELEMENT nombre\_autor(nombre, apellido)>**

**<!ELEMENT nombre (#PCDATA)>**

**<!ELEMENT apellido (#PCDATA)>**

**<!ELEMENT datos\_nacimiento(nacionalidad, ciudad, fecha\_nacimiento)>**

**<!ELEMENT nacionalidad (#PCDATA)>**

**<!ELEMENT ciudad (#PCDATA)>**

**<!ELEMENT fecha\_nacimiento (dia, mes, anio)>**

**<!ELEMENT dia (#PCDATA)>**

**<!ELEMENT mes (#PCDATA)>**

**<!ELEMENT anio (#PCDATA)>**

**<!ELEMENT institucion\_estudios (#PCDATA)>**

**<!ATTLIST Autor fotografia CDATA #IMPLIED  
id\_autor ID #REQUIRED >**

**Conjunto de Entidades: Editorial**

**Atributos:**

- id\_editorial (llave | 1:1)
- observaciones (monovalorado | 1:1)
- nombre\_editorial (monovalorado | 1:1)
- e\_mail (multivalorado | 1:n)

- telefono (multivalorado | 1:n)
- direccion (compuesto por los atributos: calle, colonia, numero, estado y codigo\_postal | 1:1)
- persona\_contacto (compuesto por los atributos: nombre y apellido | 1:1)

Para el atributo llave id\_editorial, se genera una definición de atributos de la siguiente manera:

**<!ATTLIST Editorial id\_editorial ID #REQUIRED >**

Los atributos monovalorados se transforman generando definiciones de elemento e incluyéndolos en la definición de elemento del conjunto de entidades Editorial como sigue:

**<!ELEMENT Editorial (nombre\_editorial, observaciones)>**  
**<!ELEMENT nombre\_editorial (#PCDATA)>**  
**<!ELEMENT observaciones (#PCDATA)>**

En el caso de los atributos multivalorados: e\_mail y teléfono, se genera una definición de elementos para cada uno y se incluyen en la definición de elemento del conjunto de entidades Editorial con el símbolo "+", de la siguiente manera:

**<!ELEMENT Editorial (e\_mail+, telefono+)>**  
**<!ELEMENT e\_mail (#PCDATA)>**  
**<!ELEMENT telefono (#PCDATA)>**

Para los atributos compuestos: direccion y persona\_contacto, se deben transformar a definiciones de tipo de elemento, de igual manera se transforman los atributos que lo componen, incluyéndolos posteriormente en la definición de elemento a la que pertenecen, quedando como sigue:

**<!ELEMENT direccion(calle, numero, colonia, estado, código\_postal)>**  
**<!ELEMENT calle (#PCDATA)>**  
**<!ELEMENT numero (#PCDATA)>**  
**<!ELEMENT colonia (#PCDATA)>**  
**<!ELEMENT estado (#PCDATA)>**  
**<!ELEMENT codigo\_postal (#PCDATA)>**

**<!ELEMENT persona\_contacto(nombre, apellido)>**

```
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
```

La definición integrada de del conjunto de entidades Editorial de la DTD, queda de la siguiente manera:

```
<!ELEMENT Editorial (nombre_editorial, observaciones, e_mail+, telefono+, direccion,
                    persona_contacto)>
<!ELEMENT nombre_editorial (#PCDATA)>
<!ELEMENT observaciones (#PCDATA)>
<!ELEMENT e_mail (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>

<!ELEMENT direccion(calle, numero, colonia, estado, código_postal)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT colonia (#PCDATA)>
<!ELEMENT estado (#PCDATA)>
<!ELEMENT codigo_postal (#PCDATA)>

<!ELEMENT persona_contacto(nombre, apellido)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ATTLIST Editorial id_editorial ID #REQUIRED >
```

## Referencias

Para la transformación de las referencias: Libro y Autor, y, Libro y Editorial, se deben agregar a la definición de atributos de la entidad Libro, dos atributos de tipo IDREF indicando la palabra reservada #REQUIRED, denominados id\_autor\_libro y id\_editorial\_libro, cuyos valores deber coincidir con algún valor de los atributos id\_autor y id\_editorial, de alguna de las instancias del conjunto de entidades Autor y Editorial respectivamente, la definición de la DTD del conjunto de entidades Libro queda de la siguiente manera:

```
<!ELEMENT Libro (titulo, genero, a_edicion, a_publicacion, num_ejemplares,
                observaciones, ubicacion, precio)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT genero (#PCDATA)>
<!ELEMENT a_edicion (#PCDATA)>
<!ELEMENT a_publicacion (#PCDATA)>
<!ELEMENT num_ejemplares (#PCDATA)>
<!ELEMENT observaciones (#PCDATA)>
```

```
<!ELEMENT ubicacion(pasillo, num_estanteria)>
<!ELEMENT pasillo (#PCDATA)>
<!ELEMENT num_estanteria (#PCDATA)>

<!ELEMENT precio(precio_adquisicion, precio_venta)>
<!ELEMENT precio_adquisicion (#PCDATA)>
<!ELEMENT precio_venta (#PCDATA)>

<!ATTLIST Libro isbn ID #REQUIRED
                num_pagina CDATA #IMPLIED
                portada CDATA #IMPLIED
                id_autor_libro IDREF #REQUIRED
                id_autor_editorial IDREF #REQUIRED>
```

### 6.3.1 DTD (Definición de Tipo de Documento) para la Base de Datos: Librería

Integrando las definiciones generadas para cada conjunto de entidades, la DTD resultante es la siguiente:

```
<!ELEMENT Librería (Libro+, Autor+, Editorial+)>

<!ELEMENT Libro (titulo, genero, a_edicion, a_publicacion, num_ejemplares,
                 observaciones, ubicacion, precio)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT genero (#PCDATA)>
<!ELEMENT a_edicion (#PCDATA)>
<!ELEMENT a_publicacion (#PCDATA)>
<!ELEMENT num_ejemplares (#PCDATA)>
<!ELEMENT observaciones (#PCDATA)>
<!ELEMENT ubicacion (pasillo, num_estanteria)>
<!ELEMENT pasillo (#PCDATA)>
<!ELEMENT num_estanteria (#PCDATA)>
<!ELEMENT precio (precio_adquisicion, precio_venta)>
<!ELEMENT precio_adquisicion (#PCDATA)>
<!ELEMENT precio_venta (#PCDATA)>
<!ATTLIST Libro isbn ID #REQUIRED
                num_pagina CDATA #IMPLIED
                portada CDATA #IMPLIED
                id_autor_libro IDREF #REQUIRED
                id_autor_editorial IDREF #REQUIRED>

<!ELEMENT Autor (nombre_autor, datos_nacimiento, observaciones,
                 institucion_estudios*)>
<!ELEMENT nombre_autor (nombre, apellido)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT datos_nacimiento (nacionalidad, ciudad, fecha_nacimiento)>
<!ELEMENT nacionalidad (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT fecha_nacimiento (dia, mes, anio)>
<!ELEMENT dia (#PCDATA)>
<!ELEMENT mes (#PCDATA)>
<!ELEMENT anio (#PCDATA)>
<!ELEMENT institucion_estudios (#PCDATA)>
<!ATTLIST Autor fotografia CDATA #IMPLIED
                id_autor ID #REQUIRED >

<!ELEMENT Editorial (nombre_editorial, observaciones, e_mail+, telefono+, direccion,
                    persona_contacto)>
<!ELEMENT nombre_editorial (#PCDATA)>
```

<!ELEMENT observaciones (#PCDATA)>  
<!ELEMENT e\_mail (#PCDATA)>  
<!ELEMENT telefono (#PCDATA)>  
<!ELEMENT direccion (calle, numero, colonia, estado, código\_postal)>  
<!ELEMENT calle (#PCDATA)>  
<!ELEMENT numero (#PCDATA)>  
<!ELEMENT colonia (#PCDATA)>  
<!ELEMENT estado (#PCDATA)>  
<!ELEMENT codigo\_postal (#PCDATA)>  
<!ELEMENT persona\_contacto (nombre, apellido)>  
<!ELEMENT nombre (#PCDATA)>  
<!ELEMENT apellido (#PCDATA)>  
<!ATTLIST Editorial id\_editorial ID #REQUIRED >



## Conclusiones

---

Actualmente, no existe un modelo formalmente aceptado para el diseño de bases de datos semi-estructurados y los que existen no ofrecen la expresividad deseada por los diseñadores, debido a que hacen la representación de los datos, a través de grafos dirigidos, donde los vértices son los datos y las aristas las relaciones; aunado a esto tienen cierto grado de complejidad para los diseñadores.

En este trabajo se presenta un modelo, propio, basado en el modelo entidad relación utilizado en el diseño de bases de datos relacionales, lo que hace de éste una herramienta sencilla de entender y aplicar. Cumple con las siguientes cualidades: expresividad, simplicidad, minimalidad y formalidad, ofreciendo un marco conceptual que incluye: entidades, atributos, relaciones y referencias, que en conjunto representan los requisitos y restricciones del problema, utilizando cada uno de éstos elementos para expresar distintos componentes de los requerimientos de un problema, dando como resultado final diagramas que son fáciles de leer y comprender; siendo por estas razones un buen modelo conceptual.

La característica principal del modelo, es que permite representar varios conceptos a través de sus componentes, expresa la relación jerárquica entre los conjuntos de entidades, a través de relaciones, representa restricciones de participación en atributos y en las entidades de una relación, el uso de atributos opcionales cubre una de las características de los datos semi-estructurados, permitiendo que las instancias de datos en los documentos sean heterogéneas. El uso de referencias para el diseño de la base de datos, contribuye a que exista menor redundancia de información en un documento.

A pesar de que hoy en día no existe una metodología formalmente aceptada para el diseño de bases de datos, ya sean estructurados o semi-estructurados, se opta por aplicar las fases de diseño conceptual, lógico y físico. Haciendo una analogía del contenido del presente trabajo y las fases de diseño, el proponer un modelo, contribuye a la fase de diseño conceptual; proporcionar las reglas de transformación para obtener la Definición de Tipo de Documento, a partir del diagrama resultante del modelado, es parte del diseño lógico ya que sirve para validar que la colección de documentos que se almacenará en la base de datos, tenga una estructura válida.

## Trabajo Futuro

La presente tesis contribuye al área de las bases de datos semi-estructurados con XML, con la propuesta de un modelo conceptual y las reglas de transformación para la obtención de la DTD correspondiente, sin embargo, se identifican los siguientes trabajos que pueden contribuir al modelado de estas bases de datos:

- Implementar algún algoritmo de normalización para bases de datos semi-estructurados, adecuarlo e implementarlo al modelo propuesto.
- Generar el conjunto de reglas de transformación para la generación de un XML Schema.
- Programar una herramienta de diseño, que facilite a los usuarios generar los diagramas. La herramienta debe incluir los elementos descritos en el modelo propuesto.

## Referencias

**AGUILAR Rosa Isela López** XMLVIEW- Mecanismo para la creación de vistas sobre documentos XML almacenados en la base de datos nativa exist. [Libro]. - México : [s.n.], 2010.

**B. Akmal Chaudhri. RASHID Awais., and ZICARI Roberto** XML Data Management: Native XML and XML-Enabled Database Systems [Book]. - [s.l.] : Addison-Wesley Professional, 2003. - 0201844524, 9780201844528.

**BARRIOS Luis Fernando Espino** Desarrollo de una Base de Datos Nativa XML [Libro]. - Costa Rica : [s.n.], 2009.

**BATINI Carlo., B. Shamkant Navathe., and CERI Stefano** Diseño Conceptual de Bases de Datos: Un Enfoque de Entidades-interrelaciones [Libro]. - España : Addison-Wesley, 1994. - 0201601206, 9780201601206.

**BLANCO Jaime** Notas Almacenamiento y Recuperación de Información [Libro]. - Venezuela : [s.n.], 2007. - 1316-6239.

**BOURRET Ronald** Consulting, writing, and research in XML and databases [Online] // XML and Databases. - 09 2005. - 09 20, 2011. - <http://www.rpbouret.com/xml/XMLAndDatabases.htm#isxmladatabase>.

**CORNEJO Deusdit** Informatizate [En línea]. - Grupo Informatizate, 27 de Noviembre de 2002. - 10 de 10 de 2012. - [http://www.informatizate.net/articulos/pdfs/bases\\_de\\_datos\\_nativas\\_en\\_xml\\_20020712.pdf](http://www.informatizate.net/articulos/pdfs/bases_de_datos_nativas_en_xml_20020712.pdf).

**HERNÁNDEZ René Amilcar Monroy** Lenguaje XML como Solución a las Bases de Datos y su Replicación [Libro]. - Guatemala : [s.n.], 2005.

**LAMARCA María Jesús Lapuente** Hipertexto: El nuevo concepto de documento en la cultura de la imagen [En línea]. - 19 de 11 de 2011. - 25 de 11 de 2011. - <http://www.hipertexto.info/documentos/indice.htm>.

**LE HÉGARET Philippe. WOOD Lauren., and ROBIE Jonathan** W3C [Online]. - Abril 07, 2004. - Febrero 16, 2012. - <http://www.w3.org/2005/03/DOM3Core-es/introduccion.html>.

**LEE Sin Yeung., LEE Mong Li., and A. Leonid Kalinichenko** Designing Good Semi-structured Databases [Book Section] // Conceptual Modeling — ER '99. - [s.l.] : Springer Berlin Heidelberg, 1999.

**M.Thomas Connolly. and E. Carolyn Begg** Database Systems: A Practical Approach to Design, Implementation, and Management [Book]. - [s.l.] : Addison-Wesley, 2005. - 0321210255, 9780321210258.

**MCHUGH Jason. ABITEBOUL Serge., GOLDMAN Roy., QUASS Dallan., and WIDOM Jennifer** Lore: A Database Management System for Semistructured Data [Journal] // SIGMOD Record. - New York, USA : ACM, 1997. - 3 : Vol. 26. - pp. pp. 54-66. - 0163-5808.

**MENDOZA Carlos Ricardo Cruz** Bases de Datos Nativas en XML sus Usos y Aplicaciones en el Web [Libro]. - México Distrito Federal : [s.n.], 2007.

**MORO Mirella M., LIPYEOW Lim and YUAN-CHI Chang.** Challenges on Modeling Hybrid XML-Relational Databases. [Book Section] // Open and Novel Issues in XML Database. - [s.l.] : IGI Global, 2009.

**MURALI Mani. DONGWON Lee and R. Richard Muntz** Semantic Data Modeling Using XML Schemas [Journal] // 20th International Conference on Conceptual Modeling. - Yokohama, Japan : Springer Berlin Heidelberg, 2001. - Vol. 2224. - 978-3-540-42866-4.

**POWELL Gavin** Beginning XML Databases [Book]. - Indianapolis, Indiana : Wiley Publishing Inc., 2007. - 978-0-471-79120-1.

**R. Goldman. and J. Widom** DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases [Journal] // In Proc. of 23rd International Conference on Very Large Data Bases. - 1997.

**RUBIO Daniel** Osmosis Latina [En línea]. - 20 de 10 de 2005. - 07 de 11 de 2011. - <http://www.osmosislatina.com/xml/basico.htm>.

**SAHUGUET Arnaud** Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask [Book Section] // The World Wide Web and Databases / book auth. Suciú Dan and Vossen Gottfried. - Pennsylvania : Springer Berlin Heidelberg, 2001. - 3-540-41826-1.

**SALMINEN Airi., and WM. Frank Tompa** Requirements for XML document database systems [Journal]. - New York, USA : ACM, 2001. - 605014. - 1-58113-432-0.

**SILBERSCHATZ Abraham** Fundamentos de Bases de Datos [Libro]. - España : Mc-Graw Hill Inc., 2006. - 8448146441.

**SOLTILLO Manuel Pérez** Bases de datos XML nativas [Libro]. - España : [s.n.], 2010.

**W. David Embley. and YIN Way Mok** Developing XML Documents with Guaranteed "Good" Properties [Journal] // In Proc. of 20th International Conference on Conceptual Modeling.. - London : Springer-Verlag, 2001. - 3-540-42866-6.

**W3Schools** w3schools.com [Online]. - Refsnes Data, Junio 01, 2012. - Junio 25, 2012. - [http://www.w3schools.com/dtd/dtd\\_intro.asp](http://www.w3schools.com/dtd/dtd_intro.asp).

**WILLIAMS Kevin., and BRUNDAGE Michael** Professional XML Databases [Book]. - Birmingham, UK : Wrox Press Limited., 2000. - 1861003587, 9781861003584.